НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
В. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Система класифікації міських звуків на основі глибоких нейронних мереж»

Виконав:
Студент IV курсу, групи КА-61
Смірнов Сергій Сергійович _____

Керівник:
доц., канд. фіз.-мат. наук Каніовська І.Ю. _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доцент, к. ф.-м. н. Буценко Ю.П. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Смірнова Сергія Сергійовича

1. Тема роботи «Система класифікації міських звуків на основі глибоких нейронних мереж», керівник роботи доцент, канд. фіз.-мат. наук Каніовська І.Ю., затверджені наказом по університету від «25» травня 2020 р. №1143-с

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи: модель класифікації, система класифікації міських звуків

4. Зміст роботи: дослідження та особливості предметної області систем класифікації звуків, процес класифікації міських звуків

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О.А., доцент		

7. Дата видачі завдання: 02.03.2020

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР	02.03.2020-10.03.2020	виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	11.03.2020-24.03.2020	виконано
3	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	25.03.2020-31.03.2020	виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	01.04.2020-15.04.2020	виконано
5	Завершення роботи над першим варіантом частини БДР	16.04.2020-29.04.2020	виконано
6	Проведення роботи над експериментальною частиною БДР	30.04.2020-19.05.2020	виконано
7	Проведення роботи над програмним продуктом	19.05.2020-26.05.2020	виконано
8	Оформлення БДР та аналіз отриманих результатів	19.05.202-26.05.2020	виконано

Студент С.С. Смірнов

Керівник І.Ю. Каніовська

РЕФЕРАТ

Дипломна робота: 108 с., 36 рис., 8 табл., 2 дод., 39 джерела.

МАШИННЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, АНСАМБЛЕВІ МОДЕЛІ, ЗАДАЧА КЛАСИФІКАЦІЇ, КЛАСИФІКАЦІЯ ЗВУКІВ.

Об’єкт дослідження – алгоритм класифікації звукових файлів.

Мета роботи – проаналізувати існуючі моделі класифікації, розробити власну систему класифікації звуків у зручному для користувача вигляді.

Використані моделі – у програмній реалізації було використано штучні згорткові мережі, які були об’єднані в ансамбль.

Отриманні результати – побудована система класифікації міських звуків, що може відносити звуковий файл до одного з 10 відомих класів з точністю 93.3%.

В рамках подальшого дослідження пропонується підвищувати точність моделі, розширити бібліотеку відомих класів, адаптувати модель для роботи із більш зашумленими даними, зменшувати період квантування для вхідних звукових файлів для використання системи як складової розумного будинку для розпізнавання звуків та їх фільтрації для вибіркової звукоізоляції.

ABSTRACT

Bachelor thesis: 108 p., 36 fig., 8 tabl., 2 append., 39 sources.

MACHINE LEARNING, CONVOLUTIONAL NEURAL NETWORKS, ENSEMBLE MODELS, CLASSIFICATION PROBLEM, SOUNDS CLASSIFICATION.

Object of study – algorithm for audio files classification.

Purpose – to analyze existing classification models, to develop own sounds classification system in a user-friendly form.

Used models – in the software implementation artificial convolution networks were used that were assembled into an ensemble.

Results – an urban sounds classification system was obtained that can assign an audio file to one of 10 known classes with an accuracy of 93.3%.

Further research proposes to improve model accuracy, expand the library of known classes, adapt the model to work with more noisy data, reduce the quantization period for input audio files to use the system as a component of a smart house for recognizing sounds and filtering them for selective sound insulation.

ЗМІСТ

ВСТУП

Класифікація звуків – це технологія, яка дозволяє комп’ютеру чи іншій обчислювальній машині розпізнати звуковий сегмент в аудіозаписі та класифікувати його, спираючись на бібліотеку відомих класів. Вона включає в себе дослідження з таких областей науки як електротехніка, інформатика, наука про дані, інженерія даних.

Системи розпізнавання та класифікації звуків поступово займають провідне місце у житті людини, так як стають посередниками між нею та світом. За допомогою таких систем людина може чути, виявляти нові організми в глибинах океану, досліджувати звуки, якими флора та фауна спілкується між собою. Проте поступово наш світ урбанізується, з’являється все більше міст, а старі розростаються до мегаполісів. Так само з’являються і нові звуки, які ми класифікуємо як міські.

Міські звуки – окрема область дослідження класифікації, адже вони мають свої особливості та властивості. Якщо мелодії природи людині приємно чути, то звуки міста не завжди. Тому потенційну систему класифікації міських звуків можна було б використовувати у системах вибіркової звукоізоляції в розумних будинках.

Тому об’єктом даної роботи є побудова та дослідження подібної системи, що може класифікувати міські звуки та яка отримує на вході звуковий фрагмент певної довжини.

Перший розділ даної роботи присвячений аналізу предметної області. В ньому буде розказано більш детально про актуальність побудови системи, а також про проблеми, з якими дослідники в даній області стикаються та як формалізується дана задача. У другому розділі описано один з найуспішніших підходів до класифікації звуків на основі глибоких нейронних мереж, їх математичне підґрунтя та методики побудови відповідних архітектур. Третій розділ присвячено побудові власної моделі, описано вхідний набір даних, на

яких модель буде натренерована, та проведено аналіз архітектури побудованої системи та отриманих результатів роботи моделі. У четвертому розділі буде приведений функціонально-вартісний аналіз роботи.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Людина класифікує аудіо сигнали весь час без свідомих зусиль. Розпізнавання голосу по телефону, вміння знаходити різницю між дзвінком у вхідні двері та дзвінком телефону – це задачі, які ми не вважаємо складними. Проблеми виникають, коли звук є або слабким, або зашумленим, або схожим до іншого. Наприклад, для мене складно розрізнити, які з дверей у великому залі тільки що зачинилися.

Є три головні мотиваційні ідеї для дослідження класифікації звуків. По-перше, було б повчально дізнатися, як саме люди роблять те, що роблять. Якби ми знали загальну систему, яку ми використовуємо для класифікації звуків, ми б були здатні краще діагностувати та лікувати слухові недуги. Дослідження, що дало б відповідь на ці запитання, має тенденцію бути більш психологічним та фізіологічним, ніж обчислювальним, проте методи, що застосовуються в комп’ютерних системах класифікації звуків можуть стати відправною точкою для дослідження системи класифікації звуків людини.

По-друге, було б чудово мати машину, що може робити те, що людина може робити зі звуками. Наприклад, лікарі прислуховуються до того, як пацієнт дихає, щоб діагностувати дихальні недуги, і якщо медична експертна система, запрограмована зі знаннями класифікації звуків, могла б зробити те ж саме, то жителям віддалених міст та сел країни могли б швидко поставити діагнози без витрат на консультацію з людиною-експертом, яка може перебувати в іншій країні та потребує транспортування. Таким же чином, експерт-автомеханік здатен діагностувати проблеми з двигуном автомобіля, прослухавши звуки, що двигун видає, коли працює некоректно. Існує багато областей, де люди-експерти використовують слух в роботі. Система класифікації звуків могла б забезпечити можливість виконувати таку роботу в

режимі віддаленої комунікації або у інших ситуаціях, де експерти були б недосяжні або дорогі.

Нарешті, система класифікації звуків має потенціал чути краще ніж людина. Якби комп’ютери могли допомогти нам сприймати звук так само як мікроскопи, телевізійні камери та прямі ефіри допомогли нам сприймати візуальний світ, то ми могли б знати набагато більше про світ, ніж ми знаємо зараз. Багато магнітофонів та систем голосової пошти забезпечують відтворення аудіозаписів зі змінною швидкістю, що дозволяє користувачеві швидко проходити легко зрозумілі або з низьким рівнем контексту розділи, й сповільнювати та уважно слухати галасливі або складні. Система класифікації звуків може бути побудована для автоматичної зміни швидкості відтворення, щоб зберегти постійну швидкість інформації. Подальші додатки для посилення звуку включають видалення шумів, розділення звуків та автоматичну транскрипцію музики, тексту, коду Морзе чи інших звуків, що використовуються для спілкування.

Як і у всіх загальних областях класифікації, класифікація звуків як дослідницька проблема повинна бути сегментована до її вирішення. Є багато менших проблем в даній області, над якими зараз працюють, і, принаймні, можна припустити, що деякі з цих систем можуть бути об'єднані в якийсь час у майбутньому для створення багатовимірної системи, корисної для загальних наукових досліджень та класифікації. Прагматично зрозуміло, що окремі проблеми класифікації звуків часто вирішуються за допомогою окремих систем, призначених для певного завдання.

Однією з найпоширеніших проблем класифікації звуків, яку слід вирішити останнім часом, є проблема класифікація міських звуків, адже процес стрімкої урбанізації світу розпочався не так давно (Рис. 1.1-1.3) і ми все більше і більше не можемо уявити собі життя без мегаполісів [1]. А великі міста та урбанізовані агломерації мають свої власні властивості та типи звуків, класифікувавши які можна вирішити окремі проблеми життя міст.

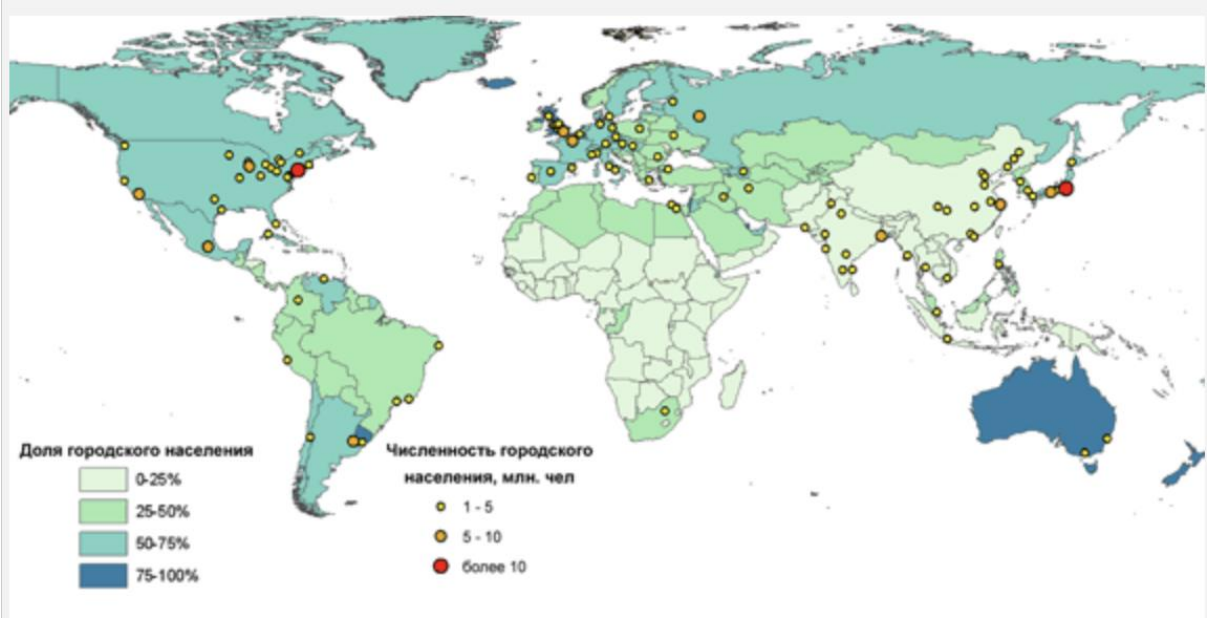


Рисунок 1.1 – Доля міського населення та міські агломерації різних розмірів, 1960 рік

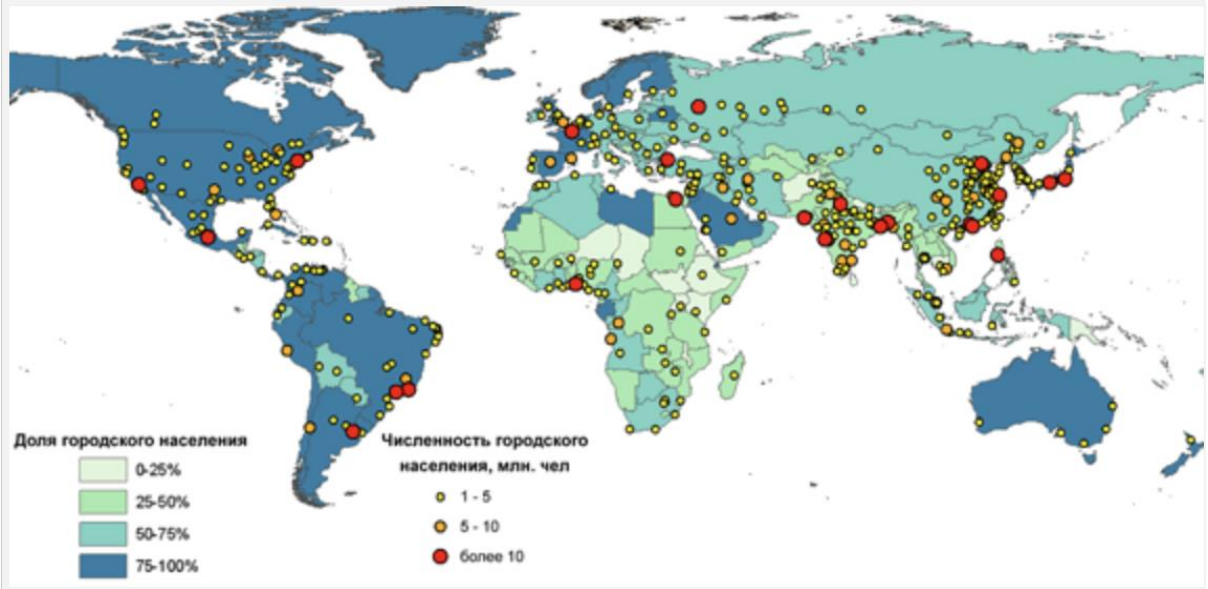


Рисунок 1.2 – Доля міського населення та міські агломерації різних розмірів, 2011 рік

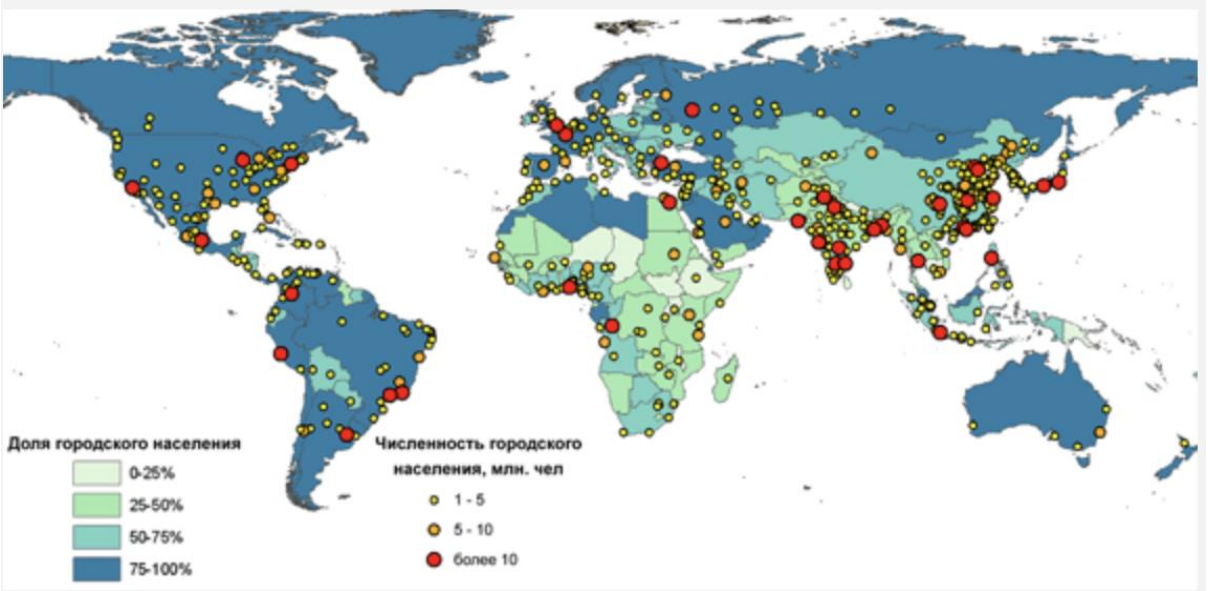


Рисунок 1.3 – Прогнозована доля міського населення та міські агломерації різних розмірів, 2025 рік

1.2 Приклади використання систем класифікації міських звуків

Одним з напрямків розвитку класифікації міських звуків є розробка аудіо підсистем для функціонування розумного будинку. Одним з таких проектів є Sweet Home - проект, спрямований на розробку нової системи розумного дому, заснованої на аудіо технології, орієнтованій на три основні аспекти: надання допомоги через природну взаємодію людина-машина (голосовий та тактильний режим комунікації), полегшення соціального включення людини у місто та забезпечення безпечного життєдіяння шляхом виявлення стресових, небезпечних чи аварійних ситуацій. Якщо ці цілі будуть досягнуті, людина зможе керувати з будь-якого місця будинку своїм оточенням в будь-який час найбільш природним для неї способом [2].

Наведемо декілька прикладів потенціального та практичного застосування систем класифікації міських звуків у проекті розумного будинку. По-перше, існує проблема так званого «голосного міста». Це означає, що

середовища міст постійно переповнені шумами та звуками, які людині не завжди хочеться чути. Недаремно все більше й більше родин купують та встановлюють у своїх квартирах та будинках системи звукоізоляції. Проте проблема в тому, що такі системи засновані на повному шумопоглинанні, а відмежуватися від абсолютно всіх звуків макросистеми міста не дуже гарна ідея. Однією з ідей є розробка розумної вибіркової системи звукоізоляції, яка б могла розпізнавати, сегментувати та класифікувати звуки, що надходять з навколишнього середовища, та віддавати в дім, тобто пропускати, лише ті, які людина попередньо у налаштуваннях системи обрала (наприклад, звук пожежної бригади, поліції чи звук ігор своєї дитини на подвір'ї пропускати, а звуки ремонтної бригади через дорогу – ні). Таким чином людина могла б створити комфортну атмосферу у своєму будинку і повністю не відмежуватись від атмосфери, що панує за дверима.

Іншою ідеєю є створення розумної системи безпеки та виявлення стресових, критичних, небезпечних чи аварійних ситуацій, що можуть трапитися як в самому будинку, так і поза його межами. Напевно кожен громадянин хотів би на свій смартфон отримувати повідомлення, коли у його дім хочуть проникнути невідомі люди та щоб дім сам викликав поліцію в таких ситуаціях. Подібним чином система могла б викликати бригаду пожежників у випадку, коли б чула специфічний звук горіння різних легкозаймистих речовин.

1.3 Основні проблеми класифікації звуків

Під час створення систем класифікації звуків в залежності від вихідних умов задачі дослідники стикаються з тими чи іншими проблемами в предметній області, які пов'язані з природою звуків. Такими проблемами є, наприклад, зашумленість даних, накладення одного звуку на інший, тощо.

Вони здатні ускладнити процес розробки системи та викликати в майбутньому неточності в моделі, якщо їм не приділяти увагу на самому початку дослідження. В даному підпункті розберемо більш докладно, яка природа цих проблем.

Перша і найголовніша проблема – це шуми в даних. З інженерної точки зору, шум – це випадковий або псевдовипадковий сигнал, який можна класифікувати за розподілом енергії в спектрі сигналу. «Білий шум» містить рівномірний розподіл енергії. Кольорові шуми містять нерівномірний розподіл. Рожевий шум має постійну потужність на октаву (спектральна щільність обернено пропорційна до частоти, тобто 1 / f) замість постійної потужності на герц. Коричневий шум має значення спектральної щільності 1 / f, де f – відповідна частота; є й інші кольорові шуми [3].

Шум також можна класифікувати перцептивно. Багато людей вважають популярну музику певної епохи «шумом». З точки зору сприйняття, шум – це звук, який неприємно чи не бажано слухати. Важко визначити цей тип шуму об'єктивно, проте деякі загальні зауваження можна зробити щодо сприйнятого шуму. Звуки, що містять великий відсоток енергії на більш високих частотах, зазвичай вважаються шумом, якщо енергія не знаходиться в гармонійно пов'язаних частинах і якщо інтенсивність сигналу порівняно висока. Ангармонічні звуки (звуки, які не містять гармонічних рядів партій), часто вважаються галасливими, знову ж таки з високою інтенсивністю, але цікаво відзначити, що деякі люди вважають білий шум, який є ангармонічним, розслабляючим для прослуховування, а також природним ангармонічні звуки, такі як океанські хвилі.

Під час дослідження цікавих для дослідників звуків, шуми можуть накладатися на основний сигнал, тому слід наперед або фільтрувати вхідний сигнал, або мати змогу його розрізняти та витісняти вже під час роботи моделі класифікації.

Проблемою може бути також накладання не лише шумів на бажані звукові сигнали, а й накладання двох чи більше основних звуків один на

одного. Тобто, коли ми, наприклад, намагаємося дослідити та класифікувати звуки співу різних птахів, не завжди вони будуть співати по черзі, а найчастіше ми будемо чути одночасний спів декількох птахів, хоч і з різних кутів від нас. У цьому випадку слід приділяти увагу сегментації звуків перед їх класифікацією, щоб два добре відомих звуки не були ідентифіковані як один і через це ми не отримали б неточність у віднесені звуку до якогось з відомих класів.

І нарешті, не завжди на вході до побудованої моделі може опинитися звук з бібліотеки відомих класів. Тому дослідники повинні мати це на увазі під час своїх досліджень та або розширити бібліотеку відомих класів новим класом, який би містив некласифіковані звукові екземпляри, або ж розробити систему, яка самостійно могла приймати рішення щодо віднесення вхідного сигналу до відомого чи раніше не баченого класу (наприклад, використовуючи байєсівську нейронну мережу [4]).

1.4 Формалізація класифікації звуків

Перш ніж перейти до аналізу існуючих підходів та методологій класифікації звуків, слід навчитися також перетворювати звуки у зручний для подальшого аналізу вигляд [5].

1.4.1 Перетворення аналогового звуку в цифровий

Цифровий сигнал – це сигнал, який представляє дані у вигляді послідовності дискретних значень; в будь-який момент часу він може приймати лише одне з кінцевої кількості значень.

Аналоговий сигнал – це неперервний на всьому проміжку часу сигнал. На протязі всього життя людина постійно чує аналогові сигнали (температура повітря, спів птахів, тощо) і вміє їх класифікувати. Для того, аби це могла зробити машина, потрібно привести такий тип сигналів до зручного для комп'ютера вигляду, тобто цифрового. Існує декілька технік для цього.

Найпоширеніша методика зміни аналогового сигналу на цифрові дані називається імпульсною модуляцією коду (PCM). Такий кодер має три основні процеси: відбір вибірок, квантування, кодування. Перед тим, як використати імпульсну модуляцію, використовують фільтр низьких частот. Він виключає високочастотні компоненти, наявні у вхідному аналоговому сигналі, щоб забезпечити відсутність у ньому перед відбором проб небажаних частотних компонентів. Це робиться для того, щоб уникнути згладжування сигналу повідомлення. Далі йде відбір вибірок. Вибірка - це процес вимірювання амплітуди сигналу безперервного часу на дискретних моментах. Існує три методи відбору: ідеальний відбір (його неможливо легко реалізувати), природний відбір проб (це практичний метод відбору проб, при якому імпульси мають кінцеву ширину, рівну T. Результатом є послідовність зразків, що зберігають форму аналогового сигналу), відбір проб плоского верху (порівняно з природним відбором цей підхід можна легко отримати. У цій методиці відбору проб верхня частина зразків залишається постійною за допомогою спеціальної схеми. Це найпоширеніший метод відбору проб). Важливе значення під час цього процесу – це частота вибірки. Згідно теореми Найквіста, частота дискретизації повинна бути щонайменше у 2 рази вищою ніж частота, що міститься в сигналі. Він також відомий як мінімальний коефіцієнт вибірки і визначається так: $Fs = 2 * fh$ [6].

Результатом відбору є серія імпульсів із значеннями амплітуди між максимальною та мінімальною амплітудами сигналу. Набір амплітуд може бути нескінченним з не цілими значеннями між двома межами. Квантування призначено для того, щоб розбити цей діапазон таких значень на скінченну кількість інтервалів [6].

Оцифровка аналогового сигналу здійснюється кодером. Після квантування кожного зразка і прийняття рішення про кількість біт на зразок, кожен зразок може бути змінений на n бітний код. Кодування також мінімізує використану пропускну здатність [6].

Оскільки імпульсна модуляція коду є дуже складною методикою, для зменшення її складності були розроблені інші методи. Найпростіша – дельта модуляція (DM). В даному підході приймає участь так званий модулятор. Модулятор використовується на стороні відправника для створення потоку бітів з аналогового сигналу. Процес фіксує невелику позитивну зміну під назвою дельта. Якщо дельта позитивна, процес записує ще 1, в іншому випадку процес записує 0. Модулятор створює другий сигнал, що нагадує сходи. Потім вхідний сигнал порівнюється з цим поступово зробленим сходовим сигналом [6].

Продуктивність дельта модулятора можна значно покращити, зробивши розмір кроку модулятора у формі, що змінюється за часом. Більший ступінчастий розмір потрібен там, коли повідомлення має крутий нахил модулюючого сигналу, і менший – коли повідомлення має невеликий нахил. Розмір адаптується відповідно до рівня вхідного сигналу. Цей метод відомий як адаптивна дельта-модуляція (ADM) [6].

1.4.2 Перехід від класифікації звуків до класифікації зображень

Після того, як машина перетворила аналоговий сигнал у цифровий, вона може вже його зберігати та обробляти. Існує багато технік для подальшого дослідження звуків: використовуючи фізичні (аналіз енергії звуків, метод нульової швидкості перетину, спектральний аналіз, аналіз основних частот, аналіз розташування формант, аналіз модуляції часових доменів) або перцептивні властивості звуків (аналіз просодії, аналіз озвучених/неозвучених

кадрів, аналіз тембрів та ритмів). В даній роботі ми зупинимося більш докладно тільки на спектральному аналізі, адже саме його ми будемо використовувати у подальшому для створення системи класифікації міських звуків.

Спектральний аналіз тримається на такому вихідному терміні як спектр сигналу. Спектр сигналу описує розподіл частот. Було показано, що людське вухо здійснює своєрідний спектральний аналіз [7], і оскільки люди витягують інформацію з спектру під час слуху, можна вважати, що система, призначена для обробки звуку, отримала б користь від вивчення спектру звуку. Крім цього, для аналізу та класифікації звуку історично використовувались спектральні методи.

Спектрограма – це мінливий за часом спектр сигналу. Для генерації спектрограми сигнал розбивається на кадри. Спектр обчислюється на кожному кадрі і ці спектри відображаються у вигляді спектру, що змінюється за часом. Результат – це міра того, як змінюється зміст частоти сигналу з часом.

Багато фізичних особливостей спектру сигналу можуть бути використані для класифікації, залежно від мети класифікації. Одним з найбільш фундаментальних спектральних заходів є пропускна здатність, яка є показником того, який діапазон частот присутній у сигналі. Ця функція використовується в [8] для розмежування мови та музики. У цьому випадку музика, як правило, має більшу пропускну здатність, ніж мова, яка не має ні низької частоти басового барабана, ні високої частоти цимбали. Ширина смуги частот також використовується в системі в [9], і в цьому випадку пропускну здатність обчислюють, беручи середнє значення різниці між частотою кожного спектрального компонента і спектральним центроїдом сигналу. Автори цієї роботи також використовують середнє значення, дисперсію та автокореляцію смуги пропускання.

Загальна ознака під назвою гармонійність використовується як особливість у кількох системах класифікації [8][9]. Гармонійність відноситься до відносин між піками спектру. Об'єкт, який вібрує резонансним чином,

наприклад, людський голос або музичний інструмент, створює звук, що має сильні пікові частоти з рівномірно розташованими інтервалами по всьому спектру. Гармонійність звуку може бути використана для розмежування між голосною і незвучною мовою або для ідентифікації музики.

Система класифікації мови / музики, представлена в роботі [10], використовує декілька особливостей, заснованих на статистичних вимірах спектру та спектрограми. До них відносяться спектральна точка відкачування, спектральний центроїд і спектральний потік. Спектральна точка перекидання – це частота, нижче якої існує більша частина спектральної енергії, і вона використовується для розрізнення голосової та незвучної мови. Спектральний центроїд – це міра середньої частоти сигналу. Музика, як правило, має більш високий спектральний центроїд, ніж мова через ударних звуків. Спектральний потік – це міра швидкості зміни спектральної інформації, і музика має тенденцію до більш високої швидкості спектрального потоку, ніж мова.

В даній роботі в якості властивості звуку, за допомогою якої він і буде досліджуватися, ми візьмемо мел частотні кепстральні коефіцієнти (MFCC). MFCC представляє собою спектральну огинаючу з коефіцієнтами, розташованими один від одного на відстані за шкалою мела, які зосереджені на частотах, що мають велике значення для людської мови і слуху. Їх використання в задачі опису характеристик фонеми зумовлене насамперед зручністю практичного застосування [11].

Мел-кепстральні коефіцієнти мають підвищену завадостійкість і дозволяють приймати достовірні рішення на відносно коротких інтервалах аналізу мови. Основною ідеєю методу мел-кепстральних коефіцієнтів є максимальне наближення інформації, що надходить на слуховий аналізатор мозку людини. Ознаки, побудовані на основі мел-кепстральних коефіцієнтів, враховують психоакустичні принципи сприйняття мови, оскільки використовують мел-шкалу, пов'язану з критичними смугами слуху [12].

Необхідно розуміти значення понять мелу і кепстра. Мел – це одиниця висоти звуку, яка заснована на сприйнятті цього звуку органами слуху людини

або, іншими словами, своєрідне уявлення енергії спектра сигналу, яке зазвичай є вектором з тринадцяти дійсних чисел. Кепстра (cepstrum) – в свою чергу, це результат дискретного косинусного перетворення від логарифма амплітудного спектра сигналу [12].

Для того щоб знайти енергію сигналу, вектор спектра сигналу перемножується з функцією вікна, в результаті чого виходить вектор коефіцієнтів. Якщо їх звести в квадрат, представити у вигляді логарифма і отримати з них кепстральних коефіцієнти, то виходять шукані мел-коефіцієнти. Кепстральні коефіцієнти можна отримати як за допомогою Фур'є-перетворення, так і за допомогою дискретного косинусоїдального перетворення. Дискретне косинусоїдальне перетворення застосовується для отримання кепстральних коефіцієнтів, воно стискає отримані результати, підвищує внесок перших коефіцієнтів і знижує внесок останніх [12].

Таким чином, можна отримати одну з дуже важливих ознак звукового сигналу, яку можна зберегти на машині у вигляді зображення (Рис. 1.4).

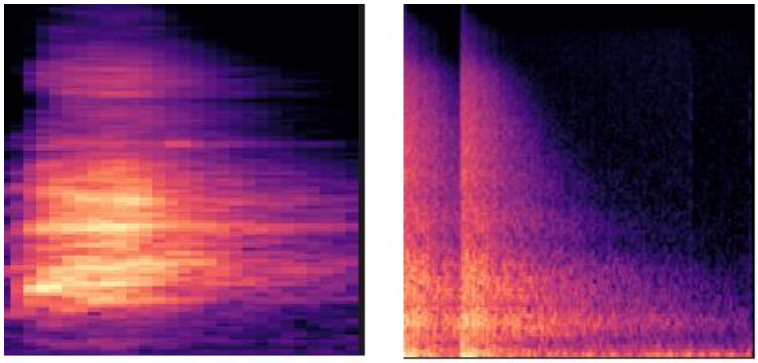


Рисунок 1.4 – Приклади спектрограмм на основі мел частотних кепстральних коефіцієнтів

Висновки

Однією з найпоширеніших проблем класифікації звуків, яку слід вирішити останнім часом, є проблема класифікація міських звуків, адже процес стрімкої урбанізації світу розпочався не так давно і ми все більше і більше не можемо уявити собі життя без мегаполісів. Побудовані системи, що можуть класифікувати міські звуки, можна буде в майбутньому використовувати, наприклад, в якості аудіо підсистем для функціонування розумних будинків (наприклад, у будинку за проектом Sweet Home). Але під час створення таких систем в залежності від вихідних умов задачі дослідники повинні приділяти достатньо уваги до проблем, з якими вони можуть зіткнутися в предметній області, які пов’язані з природою звуків. Такими проблемами є, наприклад, зашумленість даних, накладення одного звуку на інший, тощо. Для того, щоб їх уникнути та власне правильно проводити аналіз звуків, слід спочатку привести звукові сигнали до зручного для обчислення вигляду для комп’ютерних машин (перевести аналоговий сигнал у цифровий) та провести попередній відбір ознак, за якими буде проводитись дослідження властивостей (провести фізичний або перцептивний підхід до витягнення ознак зі звуків). Маючи підготовлений набір ознак у вигляді, наприклад, зображень зі спектрограмамми, побудованими на основі мел частотних кепстральних коефіцієнтах, можна проводити подальший аналіз і власне створювати системи класифікації міських звуків.

РОЗДІЛ 2
ПІДХОДИ ДЛЯ КЛАСИФІКАЦІЇ ЗВУКІВ НА ОСНОВІ ГЛИБОКОГО МАШИННОГО НАВЧАННЯ

2.1 Чому глибоке навчання?

Штучний інтелект – симуляція людської здатності до мислення, яку проводять комп’ютерні системи і яка включає в себе навчання, виявлення причинно-наслідкових зв’язків та самовиправлення. Або менш формально: система, що створена людиною, для роботи над задачами, які потребують логічного мислення та розуму.

Раніше декілька проектів з розробки штучного інтелекту намагалися бути чітко закодованими та формалізованими, використовуючи логічне виведення. Такий підхід більш відомий, як підхід баз знань у створенні штучного інтелекту [13]. Але жоден з цих проектів не зазнав великого успіху, адже неможливо людині виявити всі правила, за якими вона ж і мислить, в ручному режимі та передати це машині; заформалізувати усі можливі випадки та прийняти рішення для кожної конкретної ситуації, що трапляється з людиною у реальному житті.

Ці труднощі, з якими стикалися стародавні системи штучного інтелекту, говорять про необхідність вміння здобувати власні знання шляхом виділення патернів із вхідних даних, тобто виділення конкретних ознак, аналізуючи які система може приймати правильні кінцеві рішення. Ця здатність відома як машинне навчання. Машинне навчання дозволяє комп’ютерам, що живляться реальними даними, витягувати з них кращі шаблони та приймати кращі рішення самостійно. Прикладом таких алгоритмів є логістична регресія, наївний байєс, модель опорних векторів, тощо [14].

Витягнення властивостей з даних або об’єктів з даних є досить трудомісткий процес. Уявімо, що поточною задачею є виявлення на зображенні усіх транспортних засобів. За одну з властивостей транспортного

засобу візьмемо наявність колес. По-перше, стикаємося з проблемою формалізації для машини, як виявляти колеса (різниця розмірів, форм, кольорів у різних моделей) автомобіля. По-друге, ми стикаємося з тим, що транспортний засіб не зобов'язаний мати 4 колеса для того, щоб його ідентифікувати (наприклад, самокати, велосипеди). І, останнє, транспортний засіб може бути взагалі гужовим засобом пересування. Не слід забувати, що при роботі з зображеннями ми маємо діло з пікселями, що теж ускладнює процес. А в задачі класифікації міських звуків існує якраз підхід, за яким звуки досліджуються за допомогою картинок (про це буде розказано в наступних підрозділах). Дану проблему може вирішити підхід, що називається репрезентативним навчанням. Це є набір технік, який дозволяє системі автоматично шукати представників для виявлення властивостей або класифікації табличних даних, що витісняє ручний збір властивостей та дозволяє машині шукати параметри для навчання та використовувати їх для вирішення специфічної задачі [15].

При проектуванні таких алгоритмів наша ціль є відокремити фактори варіації, тобто ті ознаки, за якими об’єкти, що аналізуються, відрізняються один від одного. На сьогоднішній день ці фактори не завжди спостерігаються безпосередньо та їх не завжди помічає алгоритм, що може впливати на модель машинного навчання. Наразі може бути достатньо важко виділити високо-абстрактні властивості з табличних даних, наприклад, акцент людини, що говорить, для задач розпізнавання мовлення, тому що це може бути виявлено лише за допомогою складного людського рівня розуміння даних. Глибоке навчання може вирішити цю проблему у репрезентативному навчанні, запровадивши представлення, яке виражається у вигляді іншого, більш простого [16].

Слід зазначити, що підхід глибокого навчання не новітня технологія, що з’явилася декілька років тому. Його історія починається з 40-их років 20 століття (Рис. 2.1). Але вважають цей підхід до навчання новим, тому що багато років він був невизнаним, а почав набирати популярність лише з 90-их

років 20 століття – початку 21 століття. Було 3 хвилі розробки глибокого навчання: підхід під назвою кібернетика (1940 – 1960 роки), під назвою коннектуанізм (1980 – 1990 роки) та поточна хвиля (2006 – наші дні), що носить назву саме глибокого навчання [17].

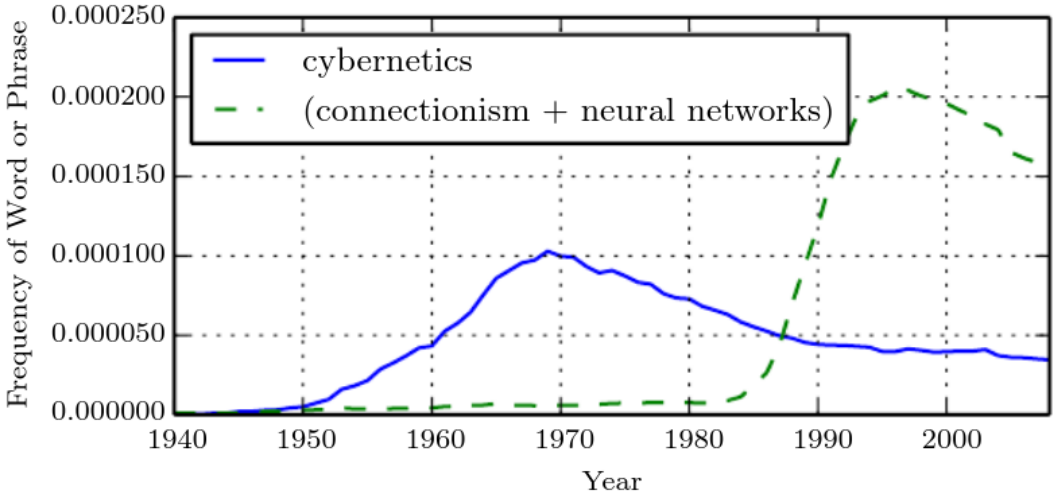


Рисунок 2.1 – Частота використання речень зі словами «Коннектуанізм» або «Нейронні мережі» та «Кібернетика» за часовими періодами, коли мова йде про підхід глибокого навчання

Але чому саме зараз глибоке навчання стало таким популярним? Однією з найголовніших причин вважається використання у цьому підході прихованих шарів (про них буде написано більш детально у наступних підрозділах). Їх може бути багато й розмір кожного окремого шару може бути великим, що дозволяє вирішувати одразу багато задач. Для правильного використання можливостей цих шарів, по-перше, треба мати великий набір даних для тренування, аби вони мали можливість витягти кожну властивість з вхідних даних. До 2000-их років науковці відхиляли глибоке навчання для вирішення своїх задач, адже або не мали великих наборів даних, або ж їм ніде було їх зберігати. З розвитком наук про Інженерію даних (Data Engineering) та Великих Даних (Big Data) ця проблема стала вирішуватися. Вони дали поштовх для подальших досліджень можливостей та потенціалу глибокого

навчання. Другим ключем успішності глибоких мереж став стрімкий розвиток комп’ютерної інженерії. Маючи навіть великий датасет, наступна проблема, з якою стикалися науковці, була неможливість швидкої обробки даних. Але на теперішній день ця проблема відійшла з виходом на ринок новітніх та швидких CPUs та GPUs. По-третє, маючи можливості використання великих об’ємів даних, точність попередніх та нових методів покращилася, що давало змогу почати використовувати їх у реальному житті, а, отже, і мати більш об’єктивні відгуки, більш ясніше поставлені бізнес цілі, та, найголовніше, зацікавленість науковців та громади у таких методах.

Оскільки дослідникам вдається досягати майже людських результатів у розпізнаванні мовлення, детекції об’єктів, розпізнаванні зображень, тощо, багато IT компаній використовують глибоке навчання для розробки своїх реальних продуктів та послуг, у тому числі і для продуктів із ідентифікації, сегментації та розпізнавання звуків.

2.2 Найпростіші моделі. Персептрони

В цьому підрозділі розкриємо суть прихованих шарів та як вони працюють на прикладі найпростіших моделей глибокого навчання.

Найпростіші моделі глибокого навчання запозичили свою ідею архітектур у людського мозку. Людський мозок має мільйони нейронів. Нейрони – це пов'язані між собою нервові клітини, які беруть участь у обробці та передачі хімічних та електричних сигналів. Дендрити – це гілки, які отримують інформацію з нейронів. Клітинне ядро або сома обробляє інформацію, отриману від дендритів. Аксон – це кабель, що використовується нейронами для відправки даних. Синапс – це зв’язок між аксонами та іншими дендритами нейронів. Таку ідею архітектури і запозичили перші вчені з питань глибокого навчання Уоррен МакКуллох та Волтер Піттс у 1943,

опублікувавши свій перший концепт ідеології штучних нейронів у відповідній статті [18].

Штучний нейрон – це математична функція, де кожен нейрон приймає вхідні дані, зважує їх окремо, сумує та проводить цю суму через нелінійну функцію активації для видачі результату. Біологічний нейрон аналогічний штучному за наступними термінами: сома – це вузол, дендрити – це вхідні дані, синапс – ваги, аксон – вихідні дані.

Наступний рисунок (Рис. 2.2) показує графічно найпростішу модель глибокого навчання – Персептрон:

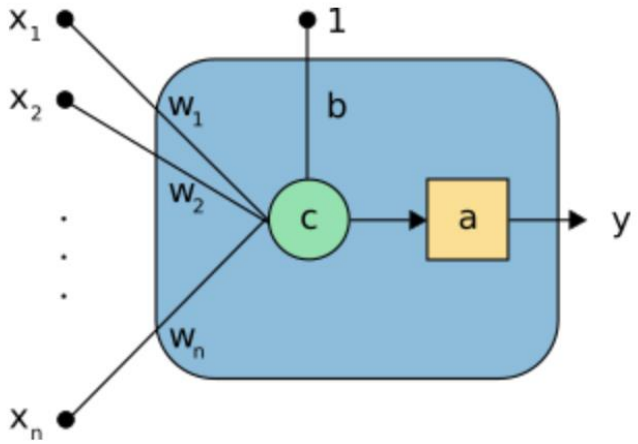


Рисунок 2.2 – Графічне представлення Персептрона

Розберемо більш докладно принцип роботи. На рисунку можна бачити такі елементи:

- вхідний вектор (x=(x1, x2, …, xn))
- вектор зміщення (b) та вектор ваг (w=(w1, w2, …, wn))
- функція комбінації (c)
- функція активації (a)
- вихідний вектор або число (y)

Вхідний вектор – це ті дані або ж вектор властивостей об’єкта, що досліджується. Зазвичай нормовані, аби при навчанні модель не звертала увагу

на різні метрики величин, що покладені в основу кожної з властивостей об’єкта дослідження.

Параметри нейрона складаються зі зміщення та вектору ваг. Зміщення це реальне число. Вектор ваг має таку ж розмірність, що й вхідний вектор.

Функція комбінації приймає вектор для отримання комбінованого значення. Зазвичай ця функція являється операцією суми. В результаті для вхідних даних x=(x_1,x_2, …, x_n) маємо поки таку проміжну функцію:

$$y = \sum_{i=1}^{n} w_i x_i + b$$

Але це ще не кінцевий вихід нейрона. Проблема функції комбінації в задачах глибокого навчання в тому, що вона лінійна. А як відомо, лише мала частина усіх процесів у реальному житті можна описати, інтерполювати чи екстраполювати за допомогою лінійних функцій. Більшість процесів у світі нелінійні, і, аби приблизити найпростішу модель до суворих реалій, використовується так звана функція активації, яка додає нелінійності у модельований процес:

$$y = a(\sum_{i=1}^{n} w_i x_i + b)$$

Існує багато функцій активацій. Найпершою була сигмоїда (Рис. 2.3), або логістична функція активації:

$$y = \frac{1}{1 + e^{-c}}$$

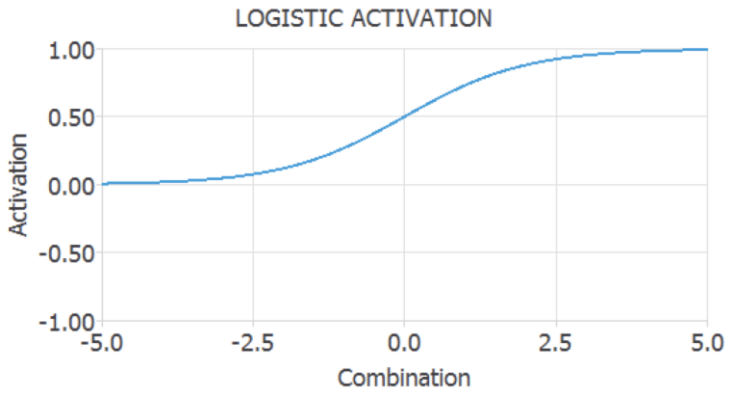


Рисунок 2.3 – Графік сигмоїди

Її перевагою є те, що її вихід можна інтерпретувати як ймовірності і найчастіше вона використовується в задачах бінарної класифікації. Невдовзі за популярністю її замінила така функція як гіперболічний тангенс (Рис. 2.4):

$$y = tanh(c)$$

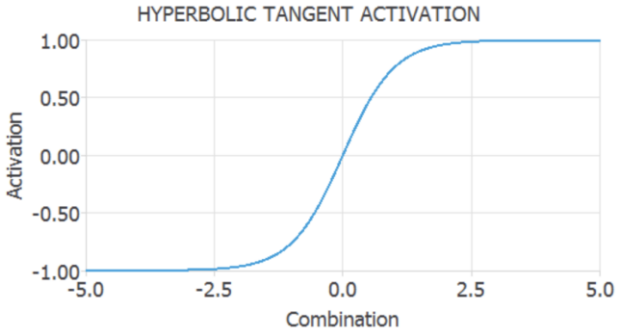


Рисунок 2.4 – Графік гіперболічного тангенсу

Однією з переваг цієї функції є те, що вона центрована. Також в останні роки набула популярності випрямлена функція лінійної активації (ReLU) (Рис. 2.5):

$$y = \begin{cases} c, \text{якщо c} > 0 \\ 0, \text{інакше} \end{cases}$$

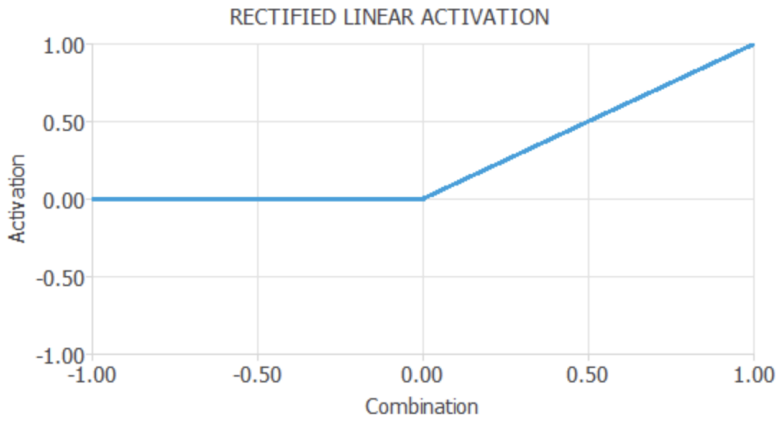


Рисунок 2.5 – Графік випрямленої функції лінійної активації

Слід зазначити, що існує ще багато модифікацій ReLU, наприклад, SELU, ELU, GELU, а також і інших функцій активації, наприклад, SoftPlus, SoftSign, Arctanh, тощо, але аналізувати кожну з них ми не будемо, адже це не головна ціль даного підпункту [19].

В результаті маємо вихідне число y, яке можна або інтерпретувати якимось чином як вирішення конкретної задачі, або ж використовувати у наступних шарах нейронної мережі. Один такий нейрон може вирішувати дуже прості задачі, але сила нейронних мереж приходить тоді, коли багато нейронів об'єднуються в шар та прив’язуються до інших подібних шарів (Рис. 2.6). На наступному малюнку можна побачити один з прикладів такої мережі:

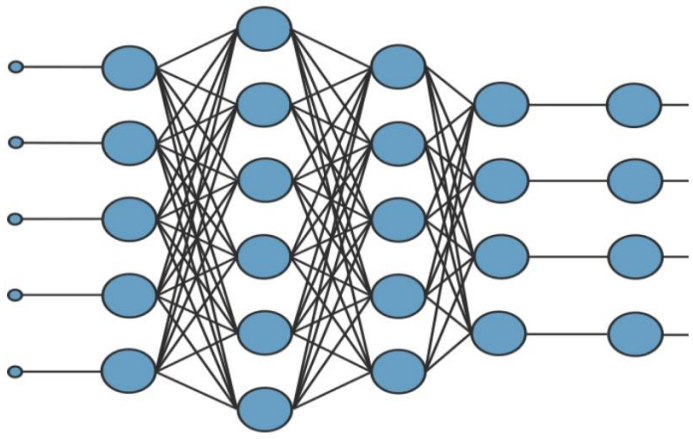


Рисунок 2.6 – Багатошарова повнозв’язна нейронна мережа з розмірністю входу 5, розмірністю виходу 4 та з 2 прихованими шарами

Така нейронна мережа називається повнозв'язною, тому що кожен нейрон приймає на вхід вектор з виходів всіх нейронів з попереднього шару. Слід зазначити, що насправді персептроном називають не один нейрон, а нейронну мережу з одним прихованим шаром. Тобто кожен нейрон самостійно може бути персептроном, але не кожен персептрон – це нейрон. Навчання такої простої моделі відбувається прогоном даних через неї та оновленням ваг та інших можливих параметрів чисельними методами та методами оптимізації, наприклад, методом оберненого розповсюдження помилки (backpropagation method). В нашій задачі класифікації міських звуків ми будемо оптимізувати наступну функцію:

$$y = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij}log(\gamma_{ij}))\,,$$

де y – вектор дійсних значень, тобто справжні класи для кожного елементу вибірки, а γ – вектор прогнозованих значень, тобто вихід моделі. Ця функція також відома як категорійна функція втрати поперечної ентропії (categorical cross entropy loss function) й найчастіше саме вона використовується в задачах мультикласової класифікації [20].

2.3 Згорткові нейронні мережі

Зображення це не що інше як матриця значень пікселів. Один із способів навчити багатошарову нейронну мережу на таких вхідних даних як зображення – це вирівняти його, тобто перетворити матрицю NxM у вектор 1x(N*M). Такий підхід в задачах класифікації може дати гарні результати, якщо зображення представляє собою базове бінарне (просте) зображення. У

випадку ж складних зображень, що мають піксельну залежність, такий метод буде не результативним, адже кожен окремий піксель не може бути представленим як окрема властивість зображення. Аби фіксувати просторові та часові залежності в зображенні та витягувати з зображення потрібний вектор властивостей для подальшого його використання в повнозв’язній нейронній мережі був розроблений метод, що базується на математичних згортках. Відповідні шари називаються згорткованими, та разом з ними для тієї ж задачі використовується шари субдискретизації або пулінга, про які буде розказано далі (Рис. 2.7).

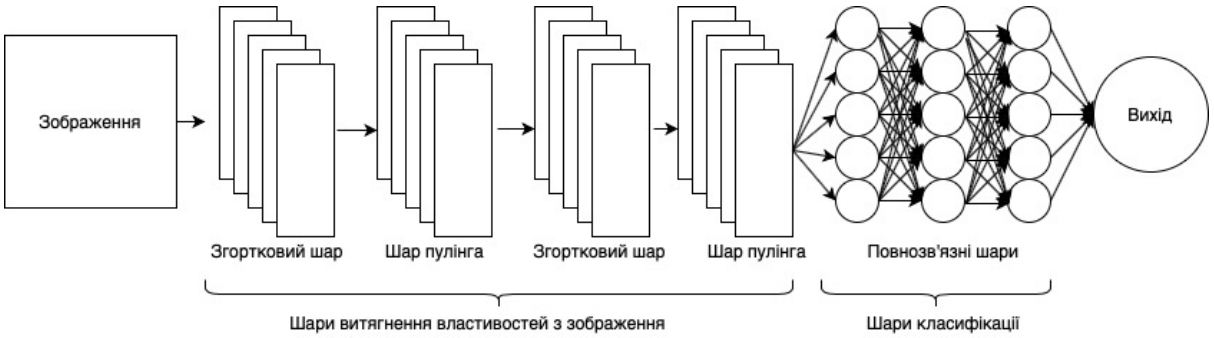


Рисунок 2.7 – Типова архітектура згорткової нейронної мережі

Роль згорткових шарів полягає в тому, щоб зменшити зображення у форму, яку легше обробляти, не втрачаючи властивостей, які є критичними для отримання хорошого прогнозу. Це важливо, коли планується створення архітектури, яка не тільки добре витягує властивості, але також масштабується до масивних наборів даних.

Сенс згорткового шару складається в тому, щоб використати математичну функцію згортки до матриці пікселів, переміщуючи відповідне вікно вздовж всього зображення. При ініціалізації ядер згортки коефіцієнти встановлюються певним випадковим чином (наприклад, використовуючи нормальне розподілення), проте під час тренування нейронної мережі вони стають тими самими фільтрами, які виділяють важливі високо-абстрактні властивості зображення, наприклад, кути об’єктів. Для багатоканального

зображення (наприклад, кольорового RGB) перед заповненням відповідної клітини вихідної матриці результати операції згортки від кожного каналу сумуються та до них додається зміщення, формуючи вихідну клітинку кінцевої матриці (Рис. 2.8). Такі згорткові шари використовуються не тільки для вхідного зображення, а й після інших згорткових шарів для вилучення більш глибоких та абстрактних властивостей [21].

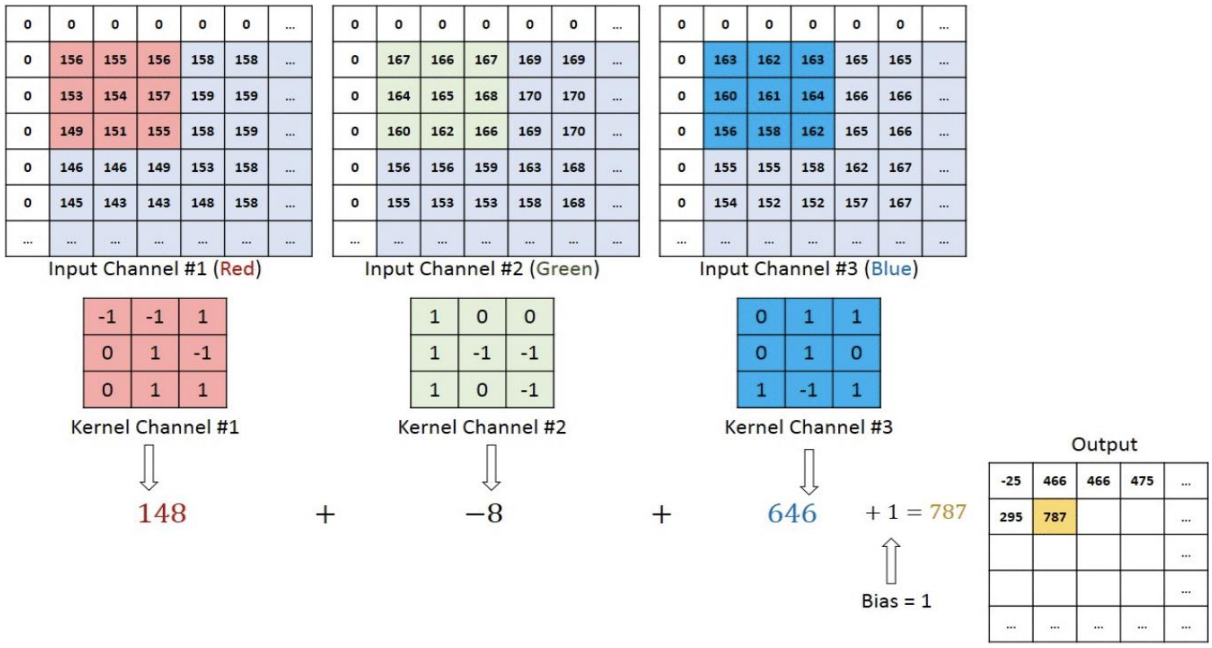


Рисунок 2.8 – Схема розрахунку вихідної матриці для згорткового шару

Подібно до згорткового шару, шар пулінгу (чи субдискретизації) відповідає за зменшення просторового розміру згорткової функції. Крім того, такий шар корисний для вилучення домінуючих рис, які є ротаційно та позиційно інваріантними, таким чином підтримуючи процес ефективної підготовки моделі. Існує 2 типи пулінгу: максимальна та середня субдискретизація (MaxPooling and AveragePooling operations). Ці операції подібні до операції згортки, але замість суми використовується операція пошуку максимуму та пошуку арифметичного середнього відповідно (Рис. 2.9).

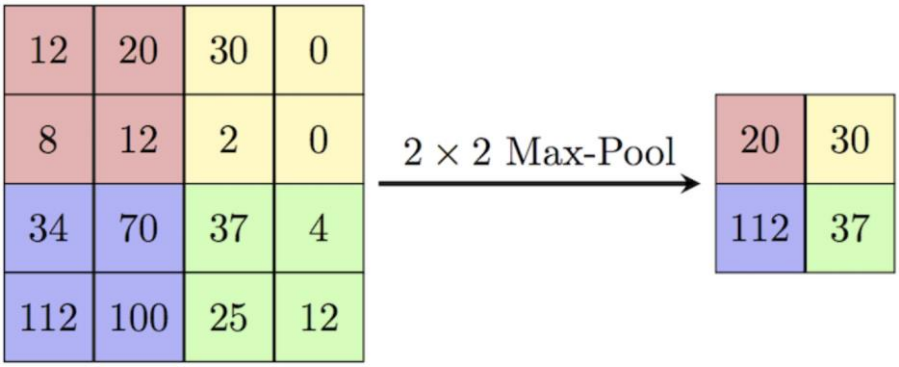


Рисунок 2.9 – Схема розрахунку вихідної матриці для шару макс пулінгу

Макс пулінг також виконує функцію шумоподавлення. Він взагалі відкидає шумні активації. З іншого боку, середнє об’єднання просто виконує зменшення розмірності як механізм придушення шуму. Отже, можна сказати, що максимальне об’єднання працює набагато краще, ніж середня субдискретизація [22].

Згортковий шар і шар пулінгу разом утворюють і-тий шар згорткової мережі. Залежно від складності зображень, кількість таких шарів може бути збільшена для фіксації деталей низького рівня ще більше, жертвуючи при цьому обчислювальною швидкістю.

Після того, як дані пройдуть через усі згорткові шари, їх можна подавати на відомі повнозв’язні шари класифікатору. При цьому на останньому шарі в задачах класифікації зазвичай використовується особлива функція активації під назвою softmax:

$$S(\mathrm{c}_i) = \frac{e^{c_i}}{\sum_j e^{c_j}}$$

Дана функція віддає на виході для кожного нейрону останнього шару мережі ймовірність приналежності зображення до відповідного класу [23]. Таким чином, об’єднавши згорткові шари, шари пулінгу, повнозв’зні шари з відповідними функціями активації можна побудувати типову згорткову нейронну мережу.

2.4 Перенавчання нейронних мереж

Перенавчання – це феномен у моделі машинного навчання та у глибоких нейронних мережах, коли побудована модель добре класифікує чи інтерполює приклади з тренувальної вибірки, проте відносно погано це робить на інших даних, наприклад, на тестовій вибірці. Причиною цього може бути занадто складна побудована модель із занадто великою кількістю параметрів для тренування. Модель, яка перенавчена, є неточною, оскільки не відображає наявну в даних реальність. А оскільки моделі глибокого навчання за замовчуванням мають велику кількість параметрів, їх дуже легко перенавчити. Мета ж будь-якої моделі – узагальнювати навчальні дані і взагалі будь-які дані проблемної області. Це дуже важливо, оскільки ми хочемо, щоб наша модель в майбутньому робила прогнози щодо даних, яких вона не бачила [24]. В даному підрозділі ми роздивимось лише три техніки, які можуть врятувати модель від даного феномену.

Перша техніка називається аугментацією. У випадку нейронних мереж, аугментація даних означає збільшення кількості зображень у тренувальній вибірці (Рис. 2.10). Деякі з популярних способів аугментації зображень це поворот, зумінг, скейлінг, шифтинг, кропінг, падінг, фліпінг, зміна яскравості, контрастності, додавання шуму [25].

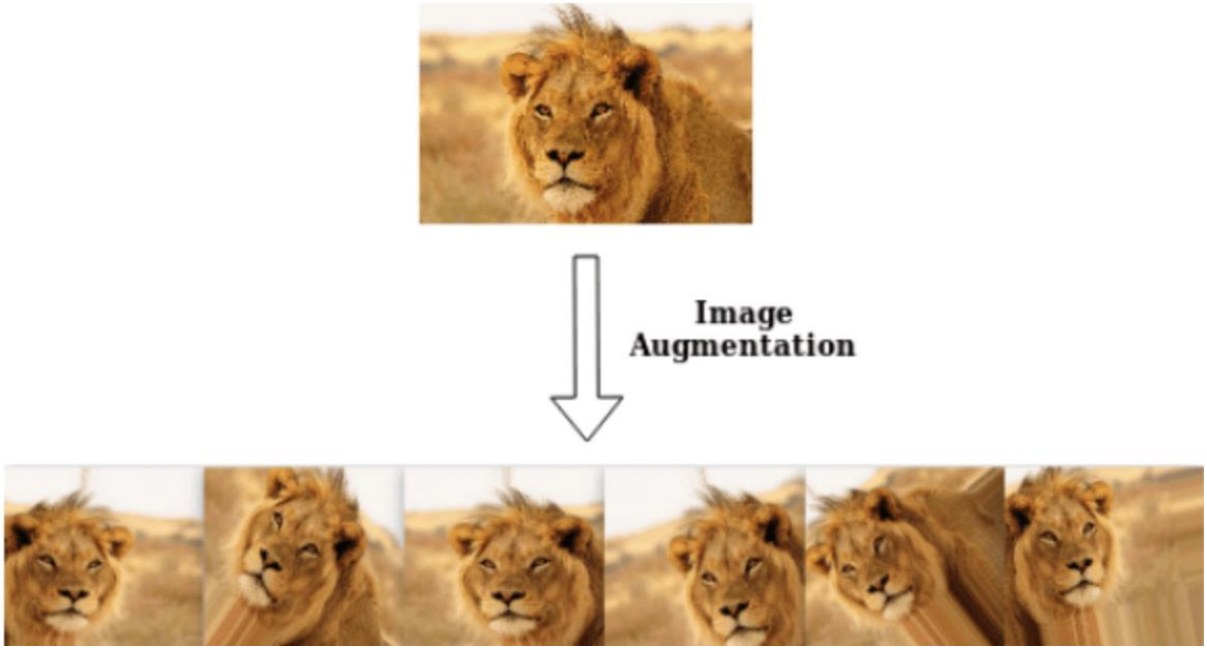


Рисунок 2.10 – Приклад аугментації вхідного зображення

Ця методика показана на малюнку вище. Як бачимо, за допомогою збільшення даних, можна створити багато подібних зображень, які можна використовувати під час тренування. Причиною уникнення перенавчання полягає в тому, що модель невзмозі тепер запам’ятати усі зразки тренувальної вибірки й мусить узагальнити параметри.

Регуляризація – це техніка зменшення складності моделі, при якій робиться додавання штрафу до функції витрат, яка оптимізується. Найпоширеніші методики: L1 і L2 регуляризація. Штраф L1 має на меті мінімізувати абсолютне значення ваг:

$$L(x,y) \equiv -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij}log(h_{ij}(\theta))) + \lambda\sum_{j=0}^{M}\sum_{i=0}^{N}|\theta_{ij}|$$

L1 регуляризація стійка до викидів та представляє собою норму у функціональному просторі L1. Штраф L2 представляє собою норму у функціональному просторі L2 та має на меті мінімізувати величину ваги в квадраті:

$$L(x,y) \equiv -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij}log(h_{ij}(\theta))) + \lambda\sum_{j=0}^{M}\sum_{i=0}^{N}\theta_{ij}^{2}$$

Яку техніку використовувати в моделі залежить від кожної конкретної задачі. Якщо дані занадто складні для точного моделювання, то L2 є кращим вибором, оскільки вона може вивчити властиві їм шаблони. L1 краще підходить для даних, які достатньо прості для точного моделювання або якщо вони дуже розріджені. Для більшості проблем комп'ютерного зору регуляризація L2 дає кращі результати [26]. Хоча також можна використовувати об'єднання цих двох технік: L1/L2 регуляризація.

Методи L1 та L2 зменшують ймовірність перенавчання шляхом модифікації функції витрат. Техніка відкидання (далі Dropout) ж змінює саму мережу (Рис. 2.11). Dropout випадковим чином відкидає нейрони з нейронної мережі під час тренування на кожній ітерації. Коли ми скидаємо різні набори нейронів, це рівнозначно тренуванню різних нейронних мереж. Різні мережі будуть перенавчатися по-різному, тому чистий ефект Dropout-у полягатиме у зменшені ймовірності перенавчання загальної моделі [27].

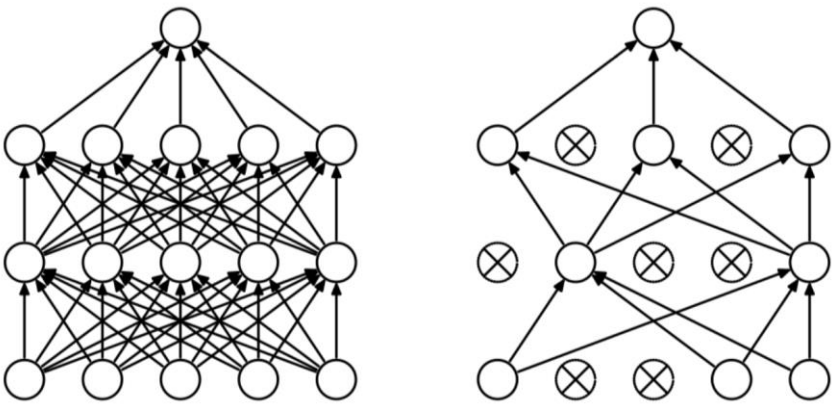


Рисунок 2.11 – Зліва – типова нейронна мережа, справа – нейронна мережа після впровадження Dropout-у на одній ітерації

2.5 Проблема зсуву та дисперсії

Під час побудови моделей прогнозування чи класифікації важливо розуміти також проблеми, що пов’язані з кінцевими помилками. Існує компроміс між здатністю моделі мінімізувати зміщення та дисперсію. Отримання правильного розуміння цих помилок може допомогти не тільки побудувати більш точну модель, а й уникнути перенавчання та недонавчання.

Зміщення – це різниця між середнім прогнозом моделі та правильним значенням, яке модель намагається передбачити. Модель з великим зміщенням приділяє дуже мало уваги даним під час навчання. Це завжди призводить до високих помилок у навчальних та тестових даних.

Дисперсія – це мінливість прогнозування моделі для даної точки даних або значення, тобто величина розкиду прогнозованих точок навколо істинних досліджуваних значень. Модель з великою дисперсією приділяє багато уваги навчальним даним і не узагальнює ті, що раніше не бачила. Як результат, такі моделі дуже добре працюють на тренувальних даних, проте мають великі помилки на тестових (Рис. 2.12).

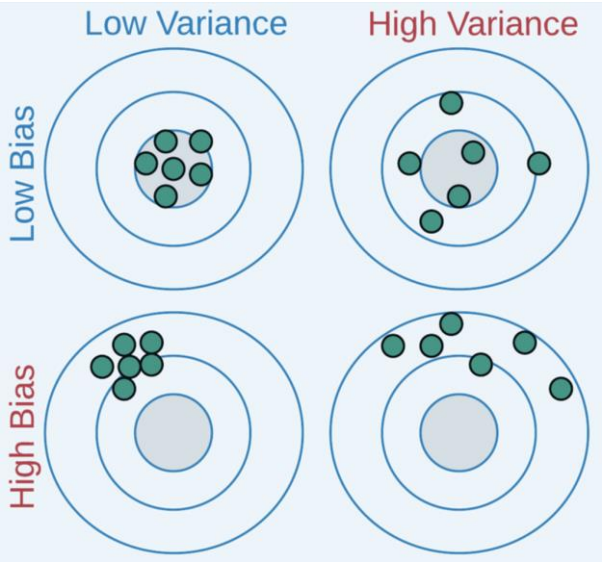


Рисунок 2.12 – Візуалізація зміщення та дисперсії за допомогою “цільової” діаграми

На наведеній діаграмі центр цілі – це область правильних значень тестової вибірки. Чим далі прогнозовані значення знаходяться від центру, тим гірша модель та більші помилки. У навчанні з вчителем, недонавчання трапляється, коли модель не в змозі захопити основний патерн даних, тобто їй не вдається дізнатися ті ознаки, якими певні класи об’єктів володіють і за якими ці класи відрізняються між собою для подальших або прогнозування, або класифікування (в залежності від задачі). Зазвичай такі моделі мають високе зміщення та малу дисперсію. Це трапляється, коли у нас малий об’єм навчальної вибірки аби побудувати точну модель або, коли ми намагаємося побудувати лінійну модель з нелінійними даними. Перенавчання трапляється тоді, коли модель фіксує разом з патернами й шуми. Це трапляється в тому випадку, коли ми тренуємо занадто багато разів нашу модель на дуже зашумленому наборі даних. Такі моделі мають велику дисперсію та низьке зміщення.

Проблема зміщення та дисперсії полягає в тому, що якщо наша модель занадто проста та має мало параметрів, вона має високе зміщення та малу дисперсію. З іншого боку, при великій кількості параметрів ми ускладнюємо модель, тим самим зменшуємо зміщення, але й збільшуємо дисперсію (Рис. 2.13). Тобто треба знайти раціональний компроміс: модель не повинна бути більш складною та менш складною ніж потрібно одночасно [28].

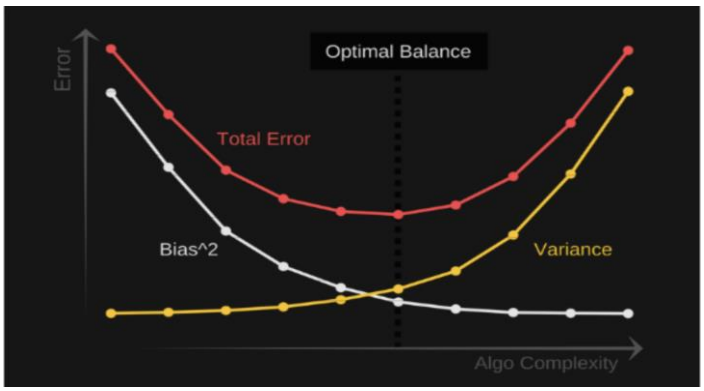


Рисунок 2.13 – Баланс між зміщенням та дисперсією для знаходження раціонального компромісу для мінімізації загальної помилки

Однією з методологій знаходження цього балансу, тобто зменшенням впливу зміщення та дисперсії на модель, це побудова ансамблю.

2.6 Ансамблеві методи

Ідея ансамблевих моделей полягає в тому, щоб взяти декілька не дуже ефективних моделей та навчити виправляти помилки один одного. Якість такої об’єднаної системи буде набагато вищою ніж кожної моделі окремо [29]. Виділяють 3 типи ансамблевих методів: беггінг, бустинг, стекінг.

Стекінг полягає в тому, щоб навчити паралельно декілька різних моделей (Рис. 2.14). Після цього їх виходи (результати навчання) разом с попередніми ознаками подаються на ще одну останню так звану метамодель, яка після навчання і приймає остаточні рішення [30].

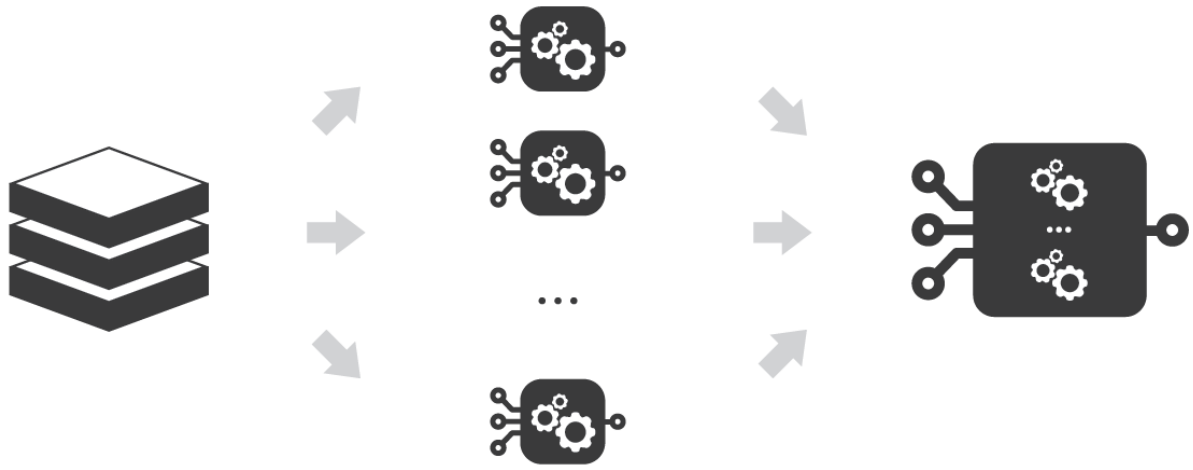


Рисунок 2.14 – Візуалізація ідеї стекінгу моделей

Ідея беггінгу полягає в тому, щоб один алгоритм багато разів навчався на випадкових вибірках з вихідного набору даних (Рис. 2.15). В кінці результати моделей усереднюються для отримання кінцевої відповіді [31].

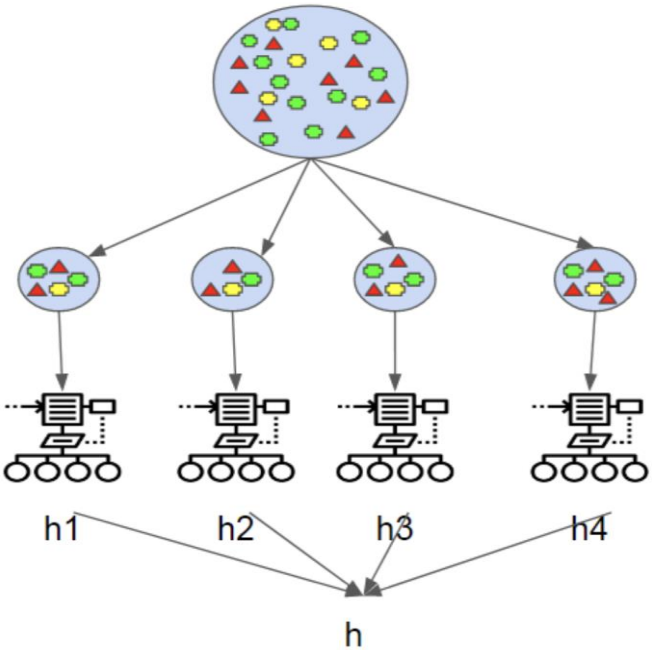


Рисунок 2.15 – Візуалізація ідеї беггінгу моделей

При бустингу алгоритми навчаються послідовно, кожен наступний приділяє особливу увагу тим випадкам, на яких попередній помилився. Як і в беггінгу, робляться випадкові вибірки з початкового набору даних, але на цей раз не зовсім випадково. За частину кожної нової вибірки береться частина тих даних, на яких попередній алгоритм відпрацював неправильно (Рис. 2.16). Тобто ми навчаємо новий алгоритм на помилках попереднього [32].

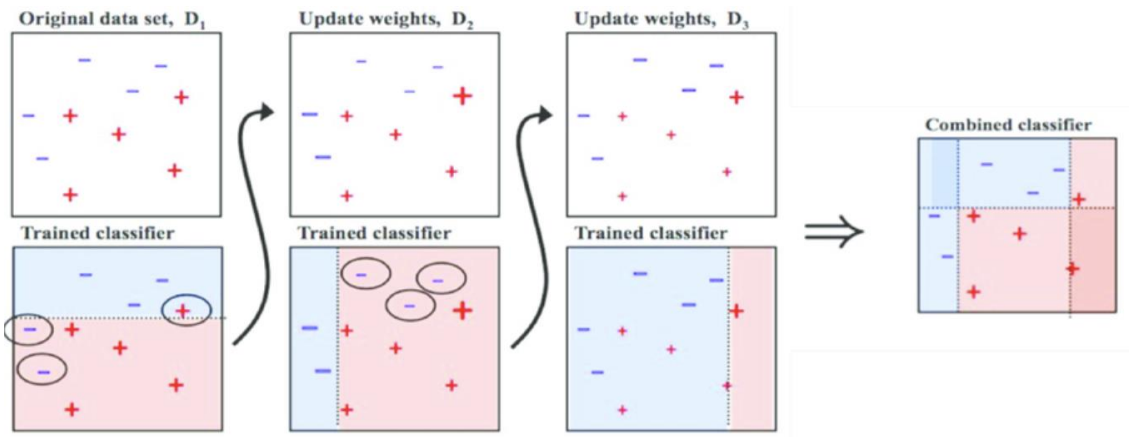


Рисунок 2.16 – Візуалізація ідеї бустингу

2.7 Огляд літератури про нейронні мережі для класифікації звуків

Більшість сучасних моделей, що класифікують звуки, засновані на методології використання глибоких нейронних мереж, особливо згорткових, так як саме такий підхід на сьогоднішній день дає найкращі результати і цьому підтвердження знаходять дослідники з усього світу у свої працях.

У статті «SoundNet: Розпізнавання звуків з нерозмічених відеофрагментів» (SoundNet: Learning Sound Representations from Unlabeled Video) [33] дослідники на основі того, що нещодавній прогрес у комп'ютерному зорі дозволив машинам розпізнавати сцени та предмети в зображеннях та відео з хорошою точністю, показують, як передати це дискримінаційне візуальне знання в звук та з хорошою точністю побудувати нейронну мережу для класифікації як природніх, так і штучних звуків (Рис. 2.17).

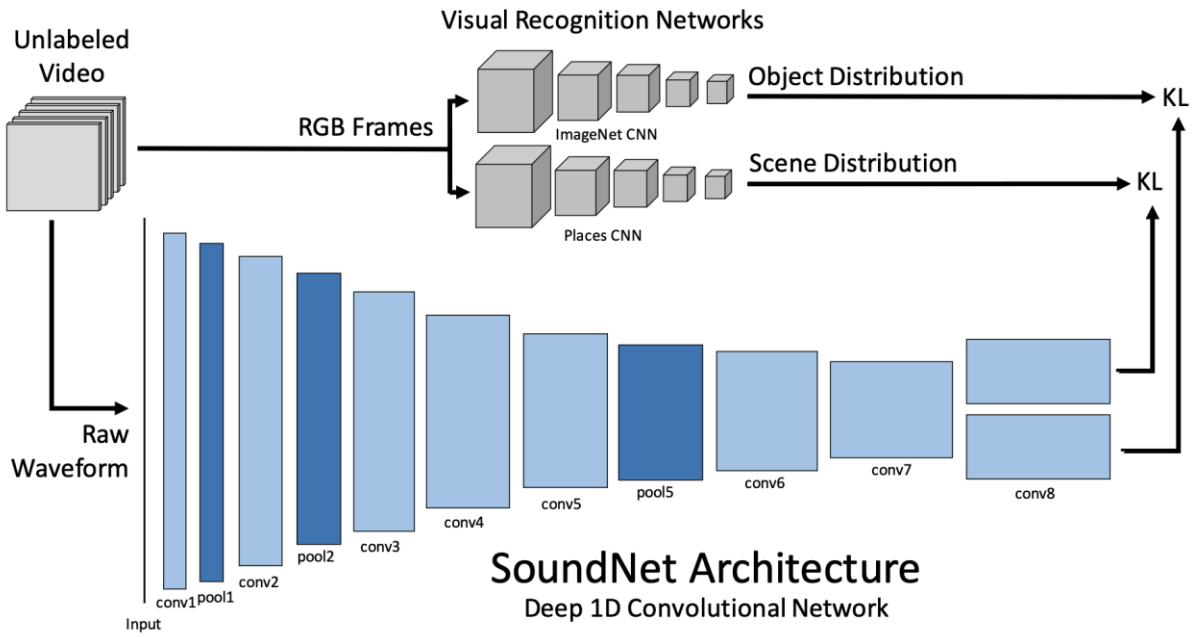


Рисунок 2.17 – Візуальне представлення нейронної мережі SoundNet для класифікації та сегментації звуків на основі відеофрагментів

У статті «ISNN: ударна звукова нейронна мережа для класифікації аудіо-візуальних об’єктів» («ISNN: Impact Sound Neural Network for Audio-Visual Object Classification») [34] дослідники пропонують свій підхід до класифікації об’єктів на відеофреймі, при чому в якості основної ознаки використовують звук. Тобто нейронна мережа для підкріплення свого рішення щодо присвоєння об’єкту на відео фрагменті до одного з відомих класів класифікує також звук, що може видавати об’єкт (Рис. 2.18).

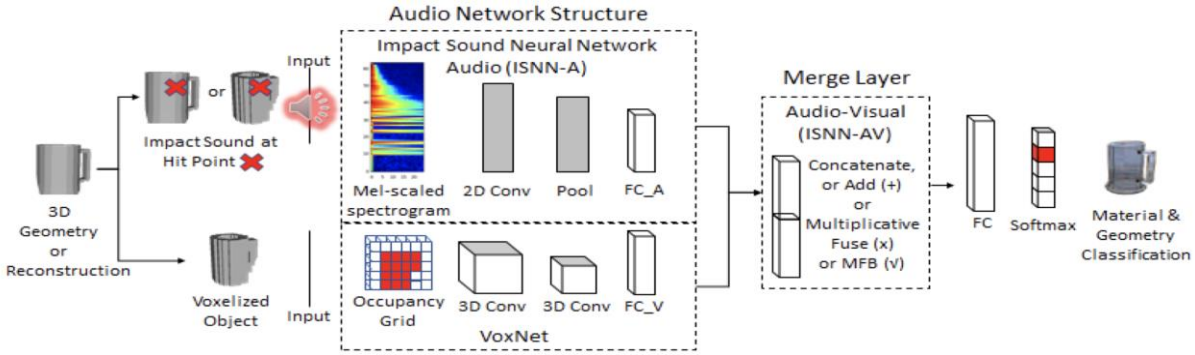


Рисунок 2.18 – Візуальне представлення нейронної мережі ISNN для класифікації та сегментації об’єктів на відео фреймах за допомогою аналізу звуків

Дослідники з університетів з Тайваню у статті «Класифікація співу птахів використовуючи згорткові нейронні мережі» («Bird Sound Classification using Convolutional Neural Networks») [35] пропонують підхід до класифікації птахів на основі їх співів, проводячи паралельно порівняльний аналіз з попередніми методами класичного машинного навчання (наприклад, дерева прийняття рішень) та рекурентними нейронними мережами.

Найцікавішою статтею, що мені трапилася на очі, була стаття Джастіна Саламона та Хуана Пабло Белло під назвою «Навчання без вчителя для класифікації міських звуків» («Unsupervised feature learning for urban sound classification») [36], що була висвітлена науковій громаді у 2015 році на міжнародній конференції з технологій обробки акустики, мовлення та сигналів від організації IEEE (International Conference on Acoustics, Speech and Signal

Processing (ICASSP)). В даній статті дослідники, спираючись на зібраний набір даних під назвою UrbanSounds8K, про який буде розказано у наступній главі, аналізують свою побудовану модель, порівнюють її точність з моделями, що пропонували інші дослідники до 2015 року, та намагаються привести свої думки щодо потенціалу використання подібних систем у сучасному світі, висвітлюючи проблеми, що пов'язані з природою міських звуків та їх класифікацією.

Висновки

На сьогоднішній день найпопулярніший і найефективніший підхід до вирішення проблеми класифікації звуків, приймаючи до уваги, що задача зводиться до задачі класифікації зображень, є використання глибоких нейронних мереж. Багато дослідників використовують такий підхід у своїх працях з аналізу та класифікації звуків. Згорткові нейронні мережі складаються з багатьох шарів, наприклад, повнозв’язного, згорткового, шарів субдискретизації. Такі моделі мають велику кількість параметрів, що може призвести до перенавчання моделей. Для уникнення цього використовують спеціальні методи, наприклад, Dropout, аугментацію, регуляризацію, тощо. Але не слід також забувати про те, що завжди треба знаходити компроміс між складністю моделі та її простотою. Однією з методологій, що дозволяє знайти такий компроміс методом зменшенням значень зміщення та дисперсії, є ансамблеві моделі, які можна використовувати і в глибокому навчанні.

РОЗДІЛ 3
РОЗРОБКА ВЛАСНОЇ МОДЕЛІ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

3.1 Аналіз даних, що були знайдені на платформі Каггл

У даній роботі будемо використовувати набір даних міських звуків UrbanSounds8K, що був знайдений на відкритій платформі Каггл. Каггл – дочірнє підприємство Google LLC – є інтернет-спільнотою науковців та практиків машинного навчання. Платформа дозволяє користувачам знаходити та публікувати набори даних, досліджувати та будувати моделі у веб-середовищі наукових даних, працювати з іншими дослідниками та інженерами машинного навчання, а також приймати участь у змаганнях з вирішення завдань із даних. Каггл розпочав свою роботу в 2010 році, пропонуючи змагання з машинного навчання, а тепер також пропонує платформу для публічних даних, облачне робоче місце для дослідників даних та освіту з штучного інтелекту.

UrbanSounds8K – відкритий набір даних. Цей набір містить 8732 розмічений звукових фрагментів протяжністю кожен не більше 4 секунд міських звуків з 10 класів:

- звук кондиціонеру (air_conditioner)
- сигнал автомобіля (car_horn)
- звуки дітей, що граються (children_playing)
- гавкіт собаки (dog_bark)
- звуки свердління (drilling)
- звук робочого двигуна (engine_idling)
- звуки пострілу з пістолету (gun_shot)
- звук відбійного молота (jackhammer)
- звуки сирени (siren)
- вулична музика (street_music)

Всі аудіо файли на платформі попередньо були відсортовані у десять кластерів, проте в даній роботі вони були переміщені між собою, щоб потенційно різний розподіл аудіо фрагментів у цих кластерах не впливав на точність майбутньої моделі. Більш детальний опис того, як набір даних був зібраний, можна знайти у джерелах [37].

До архіву файлів у розширенні .wav також прив’язується файл з метаданими архіву у розширені .xlsx. На рисунку 3.1 представлені перші рядки цього файлу.

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

Рисунок 3.1 – Перші рядки файлу з метаданими архіву UrbanSounds8K

У цьому файлі:

- slice_file_name – назва аудіофайлу. Назва має наступний формат: [fsID]–[classID]–[occurrenceID]–[sliceID].wav, де:
    1) [fsID] – Freesound ID запису, з якого взято цей уривок (Freesound – відкрита платформа для зберігання аудіо файлів).
    2) [classID] – числовий ідентифікатор класу звуку (див. опис classID нижче для отримання додаткової інформації).
    3) [occurrenceID] – числовий ідентифікатор для розрізнення різних входжень звуку в межах початкового запису.
    4) [sliceID] – числовий ідентифікатор для розрізнення різних фрагментів, взятих з одного і того ж початкового запису.
- fsID – Freesound ID запису, з якого взято цей уривок.

- start – час початку фрагмента в оригінальному записі Freesound у секундах.
- end – час закінчення фрагмента в оригінальному записі Freesound у секундах.
- salience – оцінка (суб'єктивна) значущості звуку (1 – передній план, 2 – фон).
- fold – номер кластеру (1–10), якому було виділено цей файл.
- classID – числовий ідентифікатор класу звуку (0 – air_conditioner, 1 – car_horn, 2 – children_playing, 3 – dog_bark, 4 – drilling, 5 – engine_drilling, 6 – gun_shot, 7 – jackhammer, 8 – siren, 9 – street_music).
- class – найменування класу (звуки кондиціонеру, звук сигналу автомобіля, звуки дітей, що граються, гавкіт собаки, звуки буріння, звуки робочого двигуна, звуки пострілу пістолету, звуки відбійного молоту, звуки сирени, вулична музика).

Розподіл кількості фрагментів з кожного класу у відсотках можна побачити на рисунку 3.2.

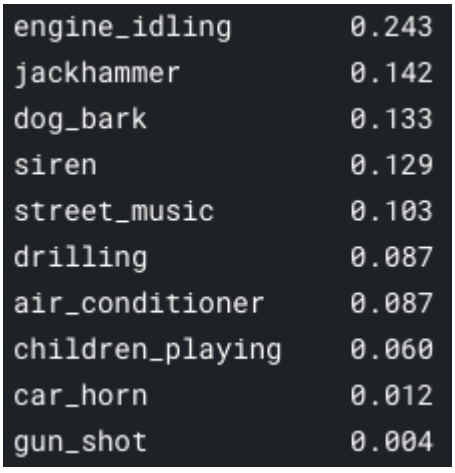


Рисунок 3.2 – Розподіл кількості фрагментів по класам у вхідному наборі даних (у відсотках)

Для того, щоб використовувати набір даних UrbanSounds8K у цій роботі, перетворимо звукові аудіо фрагменти у зображення за допомогою мел частотних кепстральних коефіцієнтів, що були описані у першому розділі цієї роботи. Для автоматичного перетворення була використана бібліотека librosa (бібліотека для аудіо та музичної попередньої обробки файлів), що написана на мові програмування python (інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією), а саме її підмодуль під назвою feature.melspectrogram. Спектрограмма мела є результатом наступних операцій:

- вхідний фрагмент розбивається на частини за допомогою ковзного вікна.
- розрахунок швидкого перетворення Фур'є для кожної частини, щоб перейти з часової області до частотної.
- весь частотних спектр розділяється на 128 рівномірно розподілених частот.
- для кожної частини величина сигналу розкладається на його компоненти, що відповідають частотам на мел шкалі.

Отримані зображення можна використовувати у подальшій роботі з згорткованими нейронними мережами (Рис. 3.3).

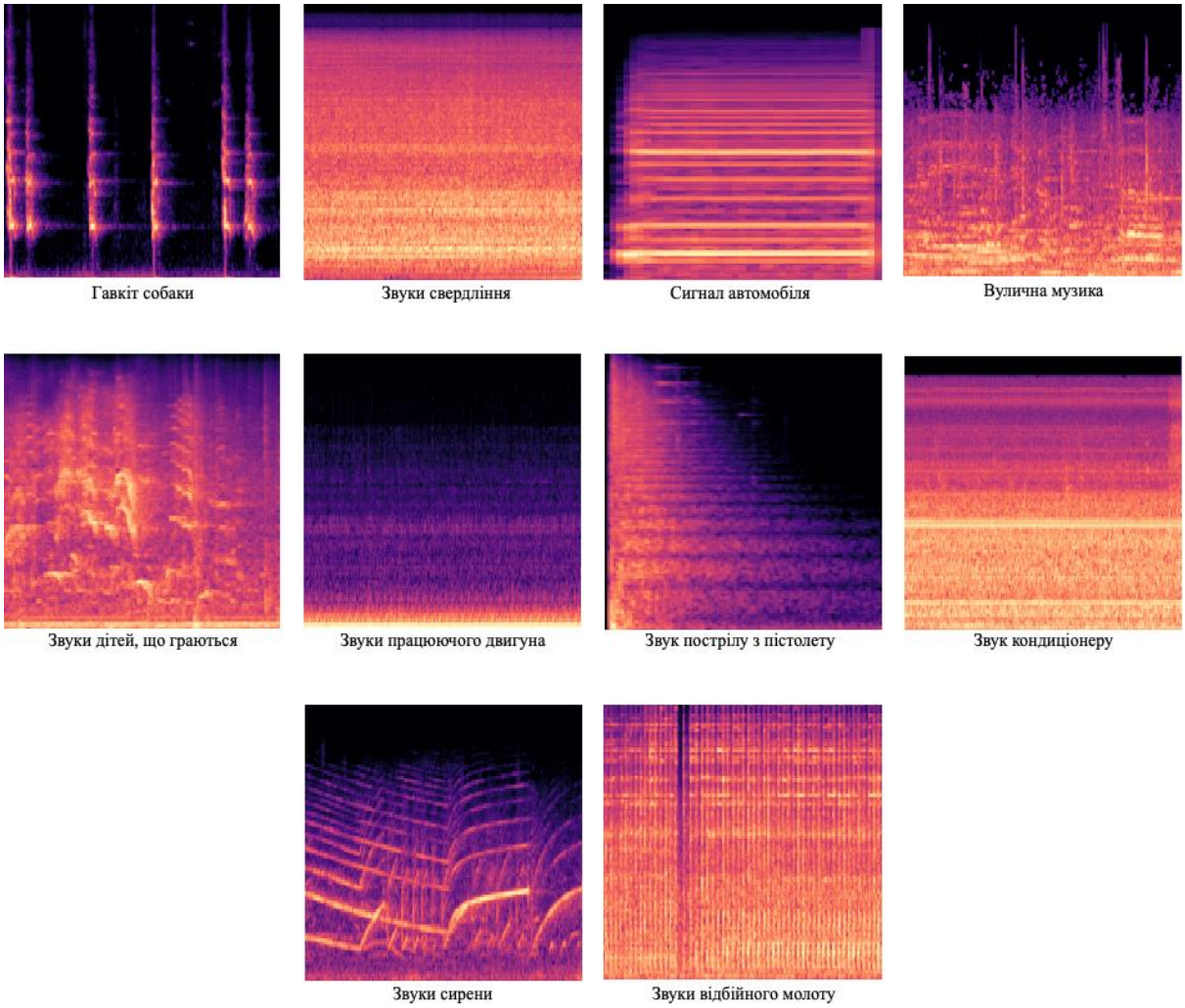


Рисунок 3.3 – Приклади мел частотних кепстральних діаграм

3.2 Архітектура побудованої моделі

В даній роботі був використаний беггінг – підхід до побудови загальної моделі, що класифікує міські звуки за вхідними мел частотними кепстральними діаграмами.

Спочатку роздивимось архітектуру однієї окремо взятої згорткової нейронної мережі у складі ансамблю. Одна така нейронна мережа складається з чотирьох згорткових шарів із кількістю та розмірністю ядер відповідно (128, 3, 3), (128, 3, 3), (64, 3, 3) та (64, 3, 3) та з п’яти повнозв’язних шарів із кількістю нейронів відповідно 512, 512, 256, 256 та 10, які йдуть після згорткових. Після

кожного згорткового шару також присутня операція субдискретизації (max-pooling операція) з ядром розмірністю (2, 2). Всі шари за функцію активацію мають гіперболічний тангенс, крім останнього. Останній шар нейронної мережі являє собою шар softmax, який дає ймовірності приналежності до кожного з відомих 10 класів для вхідного зображення. Варто також зазначити, що у кожному згортковому та у кожному повнозв'язному крім двох останніх шарах додана функція dropout з параметром 0.25. Вона використовується для того, щоб модель не перенавчалася. Її сенс полягає в тому, що кожен нейрон в шарі з ймовірністю 0.25 не приймає участь під час тренування на заданій епосі, тобто обнуляється (Рис. 3.4).

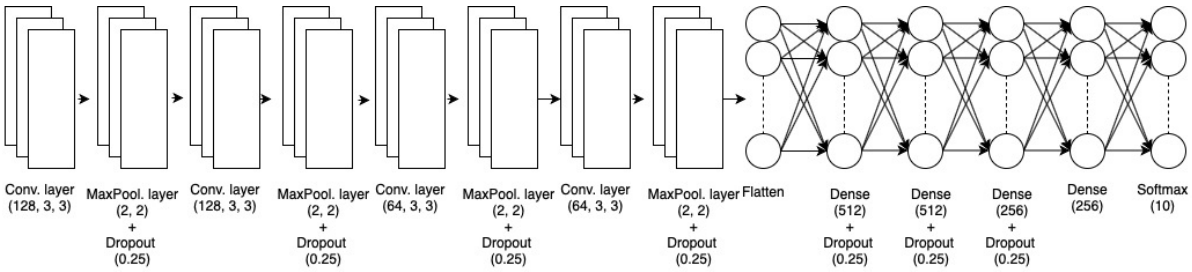


Рисунок 3.4 – Архітектура однієї нейронної мережі

В якості ансамблю було взято 5 нейронних мереж, архітектури кожної з яких аналогічні описаній раніше. Тренуються вони окремо одна від одної. Після того, як кожна нейронна мережа віддала ймовірності приналежності до кожного класу, ці ймовірності додаються. Тобто, наприклад, перша мережа повернула ймовірність 0.25, що вхідне зображення належить до першого класу, та 0.75 – що до другого. Друга нейронна мережа у свою чергу повернула 0.5 і 0.5 ймовірності відповідно. Підсумувавши, отримаємо 1.25, що вхідне зображення належить до першого класу, і 0.75 – до другого. Варто зазначити, що результуючі суми в даному випадку можуть бути більшими за 1, оскільки не нормуються. В результаті ансамбль віднесе зображення в той клас, який має найбільшу таким чином пораховану суму (Рис. 3.5).

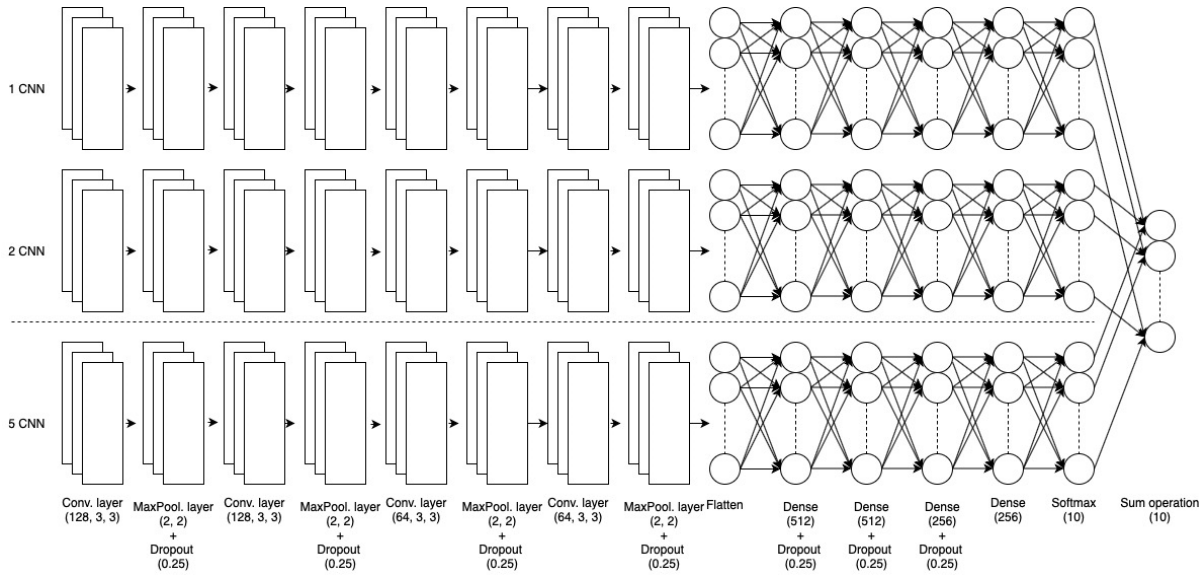


Рисунок 3.5 – Архітектура ансамблю нейронних мереж

Початковий набір даних був розбит на тренувальну та тестову вибірки випадковим чином (80% датасета тренувальна, 20% – тестова). Для нейронних мереж, які входять до ансамблю, був використаний принцип бутсрап агрегації. Тобто кожна нейронна мережа випадковим чином із загальної тренувальної вибірки обирає собі підвибірку (у даній роботі це 50: від усієї тренувальної вибірки). Далі ділить її на тренувальну та валідаційну вибірки (75%:25%) і навчається.

3.3 Аналіз результатів

Перед тим, як перейти до аналізу результатів, слід розібратися, які метрики існують та за якими буде оцінюватися модель.

3.3.1 Класичні метрики для задачі класифікації

Спочатку роздивимось метрики для бінарної класифікації даних, а потім покажемо як вони узагальнюються на мультикласові предметні області.

Найлегша для розуміння метрика оцінки результативності моделі класифікації - це точність (accuracy). Формула для точності наступна:

$$accuracy = \frac{a}{b},$$

де a – кількість правильно класифікованих елементів;

b – загальна кількість елементів.

Коли ми маємо діло з бінарною класифікацією, ми намагаємось відповісти на бінарні питання, наприклад, чи заданий звук являється пострілом з пістолету. Тому в доповнення до цього загального відношення нас цікавить також інформація по фрагментам, які були неправильно класифіковані як постріл, коли це не так (FP = False Positive), та які виявилися для моделі пострілами, хоча насправді це щось інше (FN = False Negative). Позначимо також правильно визначені фрагменти пострілів за TP = True Positive та правильно визначені інші звуки за TN = True Negative. Тоді відповідна формула точності має наступний вигляд:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Відповідні значення кількостей прогнозованих елементів (TP, TN, FN, FP) прийнято окремо виводити також у спеціальній матриці плутанини (confusion matrix) (Рис. 3.6).

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Рисунок 3.6 – Приклад оформлення confusion matrix, де у відповідних ячейках виводиться кількість спрогнозованих елементів, які підпадають під задані критерії

Перед тим, як описати сенс наступних двох метрик, слід розуміти одне з ключових понять математичної статистики: помилка першого та другого роду. Ці терміни, які властиві не лише проблемам класифікації машинного навчання, є надзвичайно важливими, коли мова йде про тестування статичної гіпотези. Помилка першого роду – хибно позитивна (False Positive) – це відмова від справжньої нульової гіпотези. Помилка другого роду – хибно негативна (False Negative) – прийняття хибної нульової гіпотези. Маючи це на увазі, ми можемо визначити іншу точність (precision) як відсоток релевантних результатів, тоді як повнота (recall) прогнозу характеризується як відсоток релевантних результатів, які правильно були класифіковані за моделюю, яка використовується. Зараз очевидно, що ці значення не дуже інтуїтивно зрозумілі, тож розглянемо кілька візуалізацій та формул, аби зрозуміти ці дві метрики (Рис. 3.7).

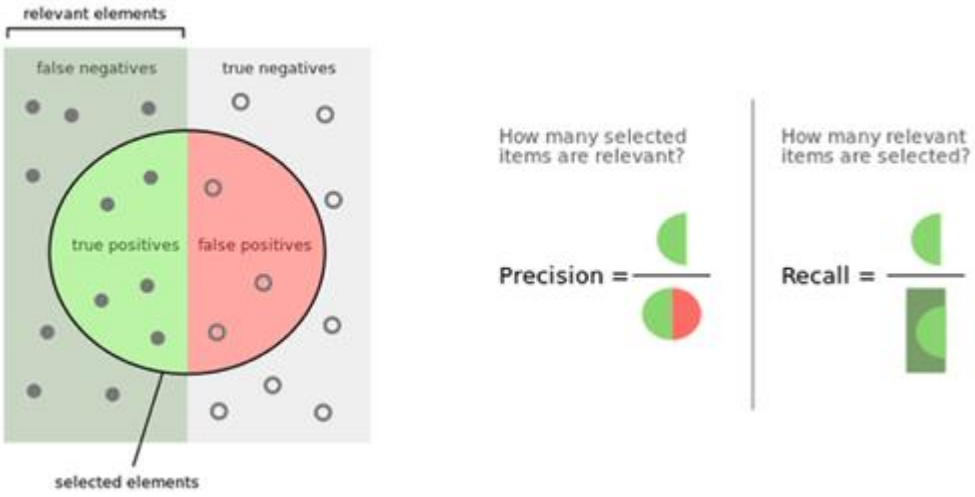


Рисунок 3.7 – Візуалізація precision та recall метрик

Коли ми говоримо про precision, ми говоримо про розподіл правильно класифікованих звуків пострілу (TP) над правильно класифікованими звуками (TP) плюс неправильно визначеними позитивними фрагментами (FP). Recall же це кількість правильно розмічених звуків (TP) над ними же плюс неправильно визначеними фрагментами як не постріли (FN):

$$precision = \frac{TP}{TP + FP},$$

$$recall = \frac{TP}{TP\ +\ FN}$$

Також є показник F1, який враховує як precision, так і recall, щоб в кінцевому рахунку виміряти точність моделі. Але яка різниця між цією метрикою та accuracy? Як було сказано на початку, помилкові позитивні і помилкові негативні класифіковані фрагменти можуть бути абсолютно вирішальними для дослідження, тоді як справжні негативні часто менш важливі для вирішення будь-якої проблеми, яку ми намагаємося вирішити, особливо в бізнес-умовах. Оцінка F1 намагається врахувати це, надаючи більшої уваги хибно класифікованим не пострілам та хибно класифікованим пострілам пістолету, не дозволяючи великій кількості правильно класифікованим не пострілам впливати на оцінку точності. F1 метрика

математично являється гармонійним середнім між recall та precision. Значення цієї метрики знаходяться в інтервалі від 0 до 1, при чому чим ближче значення до 1, тим краща модель [38].

$$f1 = 2 * \frac{precision * recall}{precision + recall}$$

Accuracy може бути легко трансформована до задачі мультикласової класифікації наступним чином:

$$accuracy = \frac{\sum_i TP_i + TN_i}{b},$$

де *i* відповідає за кожний окремий клас в предметній області.

Тобто в даній формулі TPi – кількість звуків, які були правильно віднесені до класу *i*, і так далі. Для метрики f1 не так очевидно, як побудувати відповідне відображення для переходу на мультикласові задачі. Існує 2 підходи: мікро- та макро усереднення метрик [39].

Мікро f1 метрика вираховується глобально над усіма класовими рішеннями:

$$micro\ precision = \frac{TP}{TP + FP} = \frac{\sum_i TP_i}{\sum_i TP_i + FP_i)},$$

$$micro\ recall = \frac{TP}{TP + FN} = \frac{\sum_i TP_i}{\sum_i (TP_i + FN_i)},$$

$$micro\ f1\ = \frac{2\ *\ micro\ precision\ *\ micro\ recall}{micro\ precision\ +\ micro\ recall}$$

Мікросередня f1-міра надає однакові ваги кожному класу і тому вважається середньою серед усіх пар фрагментів / класів. Як правило, такий підхід використовується у збалансованих наборах даних.

У макросприйнятті f1-міра обчислюється спочатку локально над кожною категорією, а потім береться середнє значення для всіх категорій. Precision і recall обчислюються для кожної категорії, а тоді обчислюється f1-міра для кожної категорії, а макросередня F-міра отримується шляхом взяття середнього значення f1-міри для кожної категорії як:

$$precision_i = \frac{TP_i}{TP_i + FP_i},$$

$$recall_i = \frac{TP_i}{TP_i + FN_i},$$

$$f1_i = \frac{2 * precision_i * recall_i}{precision_i + recall_i},$$

$$macro\ f1 = \frac{\sum_i f1_i}{M},$$

де M – загальна кількість класів.

Макросередня F-міра надає однакові ваги кожній категорії, незалежно від її частоти. Вона більше використовується для класифікаторів з наборами тренувальних даних, де присутні рідкісні категорії.

3.3.2 Аналіз результатів

Після проведення декількох десятків випробувань, було виявлено, що оптимальна кількість епох для тренування однієї нейронної мережі з ансамблю з архітектурою, описаною вище, є 30. Такий висновок був зроблений виходячи з того, що при більшій кількості епох моделі починають перенавчатися, тобто accuracy та значення функції витрат (за якою проводиться мінімізація та відповідна оптимізація параметрів моделей), на навчальній та валідаційній починають відрізнятися, а саме, метрики на навчальній стають кращими при

тому, що на валідаційному наборі вони залишаються або майже незмінними, або навіть і навпаки погіршуються.

Наведемо результуючі метрики, за якими можна оцінити точність моделі, на тестовій вибірці, яка була обрана способом, описаним у попередньому підрозділі. У таблиці нижче наведено середні $precision_i$, $recall_i$, $f1_i$ за термінами, описаними вище, де під середніми мається на увазі, що було проведено декілька випробувань (тренувань) з випадковим розділенням набору даних на навчальний та тестовий, а метрики, які вийшли після кожного випробування, були усереднені між собою для подачі їх в таблиці 3.1.

Таблиця 3.1 – Метрики моделі на тестовій вибірці

Клас звуку	$precision_i$	$recall_i$	$f1_i$
air_conditioner	0.84	0.98	0.91
car_horn	1	0.94	0.97
children_playing	0.91	0.91	0.91
dog_bark	0.96	0.91	0.93
drilling	0.97	0.9	0.93
engine_drilling	0.99	0.93	0.96
gun_shot	1	0.98	0.99
jackhammer	0.93	0.96	0.94
siren	0.86	0.99	0.92
street_music	0.95	0.84	0.89

Як бачимо, метрики для кожного класу за значенням більші за 0.84, а іноді доходять і до 1, що дає змогу сказати, що побудована модель достатньо точно класифікує звуки кожного класу. Найлегшим звуком для ансамблю був звук пострілу з пістолету, адже модель пропускала лише 2 фрагменти зі 100 під час класифікації (0.98 recall), та всі звуки, що були віднесені до цього класу дійсно були пострілами (precision). Найважчим же був звук вуличної музики (0.95 precision проти 0.84 recall). Таку поведінку можна пояснити наступним чином: постріл пістолету (та і взагалі й інші звуки, що були найкраще класифіковані) мають свою специфіку та достатньо низьку варіацію прикладів. Вулична ж музика на противагу, має високу варіацію прикладів, тобто під цим поняттям ми розуміємо як і хіп-хоп стиль, що передає якась радіостанція через спеціальні звукові пристрої на вулицю міст, так і легкий рок-стиль, що лунає на портативній колонці якогось хлопця, що може проходити повз записуючого звуки пристрою, що збирав набір даних UrbanSounds8K.

Наведемо тепер показники, за якими можна оцінити модель в цілому, не виділяючи класи між собою, тобто *accuracy, micro* and *macro f1* показники, в таблиці 3.2.

Таблиця 3.2 – Метрики моделі на тестовій вибірці

accuracy	*micro f1*	*macro f1*
0.93	0.93	0.94

Як бачимо, мікро та макро усередненні показники f1 достатньо близькі між собою. Це означає, що або в даних не було дисбалансу класів (розподіл фрагментів на тренувальній вибірці по класам був близьким до рівномірного), або модель, незважаючи на цю проблему, змогла натренуватися на заданих фрагментах достатньо добре, щоб класифікувати тестові фрагменти з високою

точністю. Accuracy моделі дорівнює 0.93, що дає змогу сказати, що 93 фрагменти зі 100 поданих на модель будуть класифіковані правильно.

Проаналізувавши метрики, модель можна описати не трійкою параметрів 0.93–0.93–0.94 (що відповідають за три метрики, які наведені у таблиці вище), а достатньо лише однією: 0.933, що є середнім між цими трьома показниками та яку можна інтерпретувати як точність ансамблю за будь-якою класичною метрикою класифікації (accuracy, micro f1, macro f1, precision для і-того класу, тощо).

3.4 Розробка системи

Для того, щоб можна було використовувати побудовану модель, було розроблено десктопний додаток на мові програмування python. Для розробки інтерфейсу було використано бібліотеку PyQt5 (набір розширень графічного фреймворку Qt для мови програмування python) (Рис.3.8).

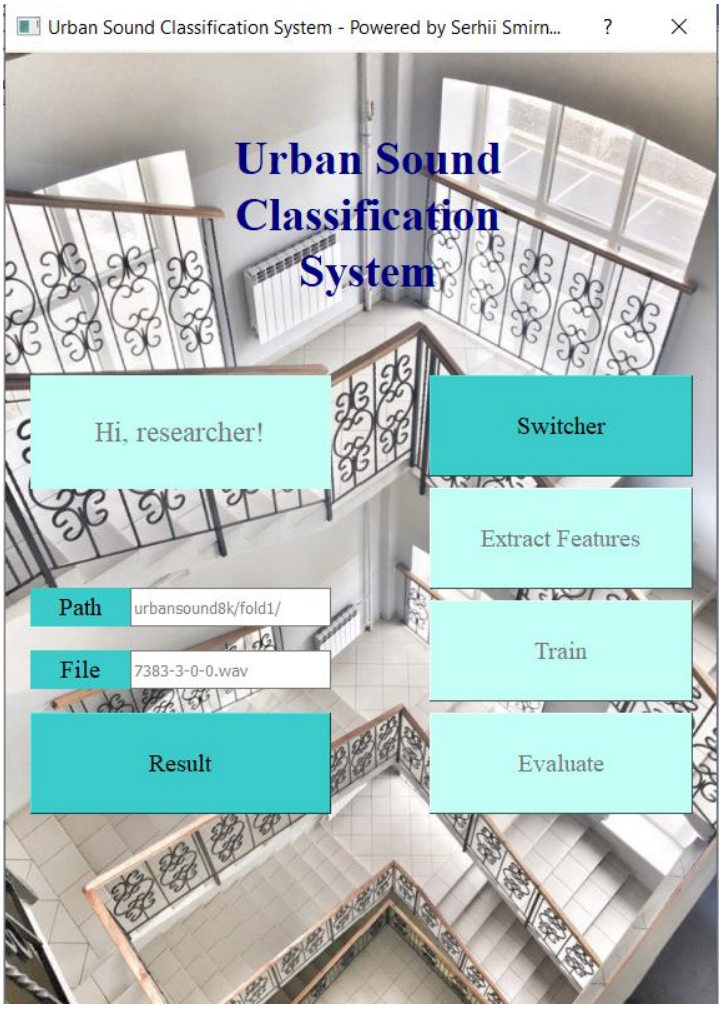


Рисунок 3.8 – Інтерфейс програми для класифікації міських звуків

При запуску програми користувач має змогу переключитися з режиму класифікації на режим тренування моделей та навпаки (кнопка Switcher). За замовчуванням програма знаходиться в режимі прогнозування, тобто класифікації. Для того, щоб класифікувати якийсь файл, треба ввести назву файла у поле File та шлях до нього на комп’ютері у полі Path і натиснути кнопку Result. Після цього у спеціальному вікні можна побачити клас, до якого модель віднесла заданий аудіо файл (Рис. 3.9).

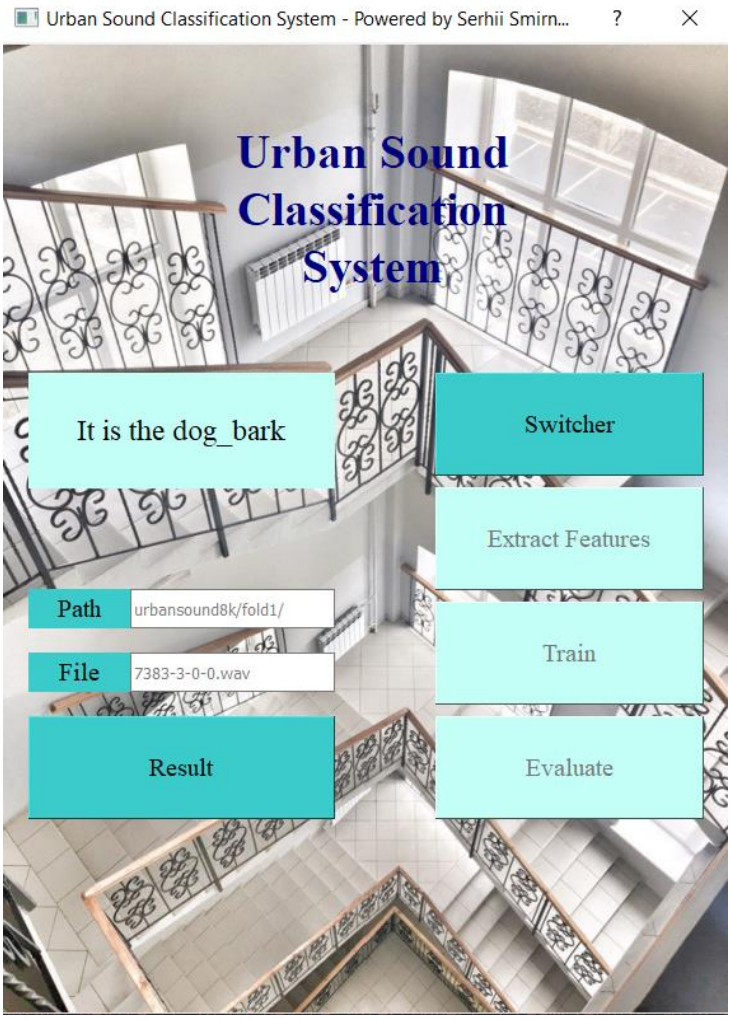


Рисунок 3.9 – Приклад роботи програми у режимі класифікування для файлу 7383-3-0-0.wav

Для того, щоб перетренувати модель, слід переключитися в режим навчання, натиснувши кнопку Switcher. Після цього користувачеві будуть доступні кнопки Extract Features (запустити процес обробки аудіо файлів, тобто побудова мел частотних кепстральних діаграм), Train (запустити процес навчання нейронних мереж) та Evaluate (оцінити ансамбль на тестовій вибірці). При цьому у спеціальному полі буде виведено точність (accuracy) побудованої моделі на тестовій вибірці. Також можна змінити деякі налаштування системи, які знаходяться у файлі config.ini. Наприклад, користувач у цьому файлі може вказати місцезнаходження архіву з даними на комп’ютері, на якому модель буде тренуватися, або кількість нейронних мереж, що повинні входити до кінцевого ансамблю.

Якщо користувач введе некорректні дані або виникне помилка програми під час роботи, у спеціальному вікні буде висвітлено відповідне повідомлення.

Висновки

При розробці системи класифікації міських звуків був використаний набір даних міських звуків UrbanSounds8K, який був знайдений на відкритій платформі Каггл. Цей набір містить 8732 розмічений звукових фрагментів протяжністю кожен не більше 4 секунд міських звуків з 10 класів. Для того, щоб класифікувати звуки, було натренеровано 5 згорткових нейронних мереж протягом 30 епох кожна, які потім були об'єднані в ансамбль за беггінг-принципом. Модель оцінювалася за декількома метриками та показала наступну трійку результатів: 0.93 за accuracy, 0.93 за micro f1 та 0.94 за macro f1, що дає змогу описувати ансамбль тільки за одним показником: 0.933, що є середнім між цими трьома показниками та за яким можна інтерпретувати точність ансамблю за будь-якою класичною метрикою класифікації.

Ці результати виявилися достатньо хорошими, щоб побудувати десктопну додаток-систему за допомогою розширення PyQt5 з інтуїтивно зрозумілим інтерфейсом на англійській мові, щоб користувач міг взаємодіяти з моделлю та класифікувати потрібні йому аудіо файли.

РОЗДІЛ 4
ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка завдання

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для класифікації міських звуків. Програмний продукт призначено для використання на персональних комп'ютерах з операційною системою Windows. Інтерфейс користувача був розроблений за допомогою мови програмування Python у середовищі розробки PyCharm.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

4.2 Обґрунтування функцій та праметрів програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу класифікації міських звуків.

Виходячи з конкретної мети, можна виділити наступні основні функції програми:

F_1 – вибір мови програмування: а) Python; б) Java;

F_2 – вибір середовища розробк: а) PyCharm; б) IntelliJ Idea.

F_3 – вибір бібліотеки для обробки даних та побудови моделі: а) Keras; б) Tensorflow;

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно- негативну матрицю варіантів основних функцій (табл. 4.1).

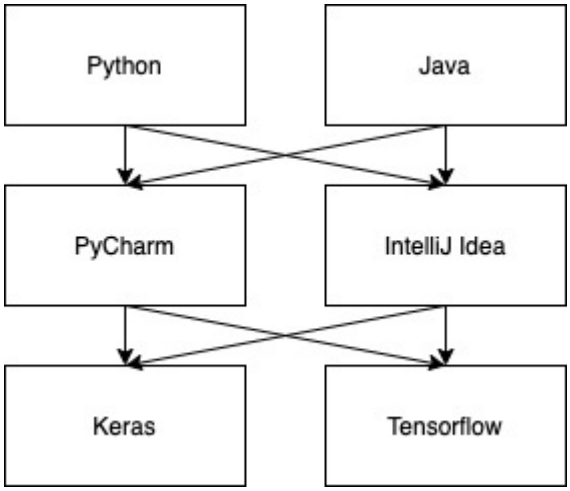


Рисунок 4.1 – Морфологічна карта

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	А	Велика кількість бібліотек, простіший програмний код	Низька швидкодія, більший час на виконання операцій
	Б	Кросплатформеність	Менша швидкість, менше модулів
F2	А	«Розумний» редактор коду, зручна навігація, швидкий та безпечний рефакторинг	Повільний
	Б	«Розумний» редактор коду	Повільний

Кінець таблиці 4.1

Основні функції	Варіанти реалізації	Переваги	Недоліки
F3	А	Безкоштовність, легкий у використанні, швидкість	Неможливість ускладнювати модель
	Б	Надійність, безкоштовність	Займає багато коду, важкий для розуміння

За наявності позитивно-негативної матриці можна робити висновки щодо доцільності використання одних варіантів та недоцільності використання інших. На основі порівняльного аналізу варіантів реалізацій основних функцій по їх перевагам та недолікам можна виключити варіанти F_1 б, F_2 б, тоді варіанти, які залишилися:

- F_1 а) — F_2 а) — F_3 а)
- F_1 а) — F_2 а) — F_3 б).

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	с	15	10	3

Кінець таблиці 4.2

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Об’єм пам’яті для збереження даних	X2	Кб	1000	300	100
Час обробки даних	X3	с	20	3	1
Потенційний об’єм програмного коду	X4	кількість рядків коду	3000	1000	500

За даними, наведеними у таблиці 4.2, будуються графічні характеристики параметрів – рис. 4.2-4.5.

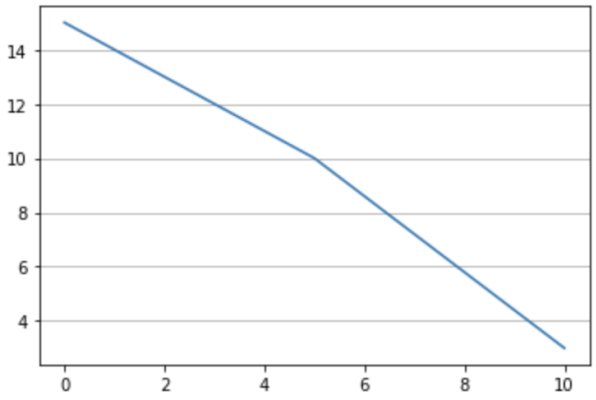


Рисунок 4.2 – Швидкодія мови програмування

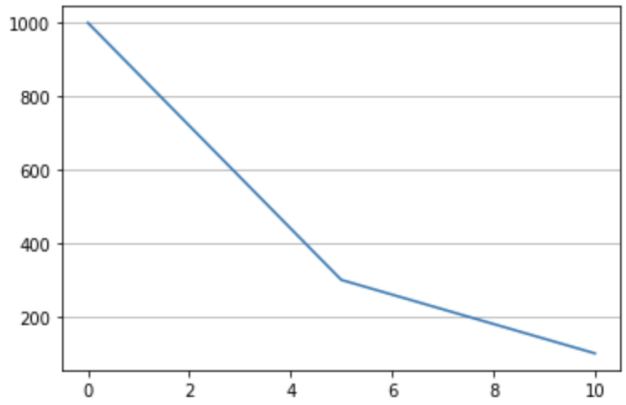


Рисунок 4.3 – Об’єм пам’яті для збереження даних

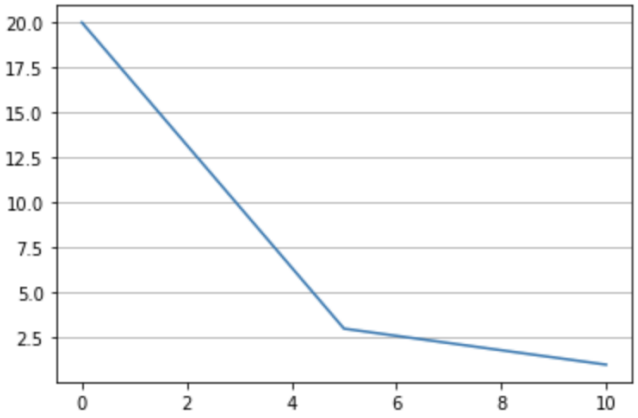


Рисунок 4.4 – Час обробки даних

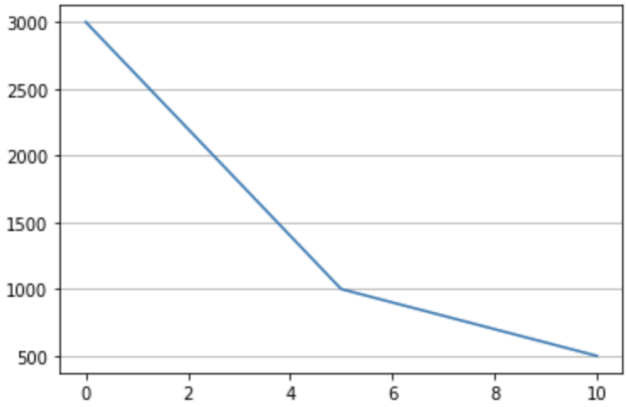


Рисунок 4.5 – Потенційний об’єм програмного коду

Вагомість параметрів визначається методом попарного їх порівняння на основі результатів ранжування експертами та попарного порівняння параметрів. Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Назва параметра	Одиниця виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
		1	2	3	4	5	6	7			
Швидкодія мови програмування	с	1	2	2	1	2	1	2	11	-6.5	42.25
Об’єм пам’яті для збереження даних	Кб	2	1	1	2	1	2	1	10	-7.5	56.25
Час обробки даних	с	4	4	3	4	4	4	3	26	8.5	72.25
Потенційний об’єм програмного коду	кількість рядків коду	3	3	4	3	3	3	4	23	5.5	30.25
Разом		10	10	10	10	10	10	10	70	0	200

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 200}{7^2(4^3 - 4)} = 0.816 > W_k = 0.67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, який дорівнює 0.67. Скориставшись результатами ранжування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4. Зазначимо, що за найбільший ранг приймається значення 4, за найменший – 1.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числова значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	>	<	>	<	>	>	1.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	<	<	<	<	<	<	<	<	0.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	>	>	<	>	>	>	<	>	1.5

Розрахунок вагомості занотуємо у таблицю 4.5, показники рівня якості у таблицю 4.6.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i^0	$K_{\text{в}i}^0$	b_i^1	$K_{\text{в}i}^1$	b_i^2	$K_{\text{в}i}^2$
X1	1.0	1.5	0.5	0.5	3.5	0.2186	12.25	0.209	44.875	0.21
X2	0.5	1.0	0.5	0.5	2.5	0.1562	9.25	0.156	34.125	0.16
X3	1.5	1.5	1.0	1.5	5.5	0.344	21.25	0.36	76.25	0.358
X4	1.5	1.5	0.5	1.0	4.5	0.2812	16.25	0.275	58	0.272
Всього:					16	1	59	1	213.25	1

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	10	5	0.21	1.05
F2(X2)	А	300	5	0.16	0.8
F3(X3)	А	1	10	0.358	3.58
	Б	3	5	0.358	1.79
F3(X4)	А	1000	5	0.272	1.36
	Б	1000	5	0.272	1.36

$$K1 = 1.05 + 0.8 + 3.58 + 1.36 = 6.79$$

$$K2 = 1.05 + 0.8 + 1.79 + 1.36 = 5$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення. Тому віддаємо перевагу саме йому.

4.3 Економічний аналіз варіантів розробки ПП

Для того, щоб визначити вартість розробки ПП, проведемо спочатку розрахунок трудомісткості. Всі варіанти включають в себе два окремих завдання:

а) Розробка моделі класифікації міських звуків

1) реалізація з бібліотекою keras

2) реалізація з бібліотекою tensorflow

б) Розробка програмної оболонки

Завдання 1 за будь-яким з двох варіантів реалізації за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1 для будь-якого варіанту реалізації (Тр=90 людино-днів, Кск=1, Кст=0.6 для 1 варіанту реалізації та 0.8 для другого); а в завданні 2 – до групи 3 (Тр=27 людино-днів, Кск=1, Кст=0.8). Для реалізації завдання 1 використовується довідкова інформація (Кп=1.7), а завдання 2 використовує інформацію у вигляді даних (Кп=0.9).

$$T1.1 = 90 * 1.7 * 0.6 = 91.8 \text{ людино} - \text{днів},$$

$$T1.2=90*1.7*0.8=122.4 \text{ людино}-\text{днів},$$

$$T2=27*0.9*0.8=19.44 \text{ людино}-\text{днів},$$

$$T_I=(91.8+19.44)*8=889.92 \text{ людино}-\text{годин},$$

$$T_{II}=(122.4+19.44)*8=1321.648 \text{ людино}-\text{годин}$$

В розробці приймають участь два інженера з даних з окладом 10000 грн. Визначимо зарплату за годину:

$$C = \frac{10000 \ + \ 10000}{2 * 21 * 8} = 59.5 \text{ грн}$$

Зарплата поваріантно:

$$C_{1зп} = 59.5 * 889.92 * 1.2 = 63540.29 \text{ грн},$$

$$C_{2зп} = 59.5 * 1321.648 * 1.2 = 94365.67 \text{ грн}$$

Відрахування на соціальний внесок становить 22%:

$$C_{1від} = 0.22 * 63540.29 = 13978.86 \text{ грн},$$

$$C_{2від} = 0.22 * 94365.67 = 18873.13 \text{ грн}$$

Визначаємо витрати на оплату однієї машино-години. З урахуванням заробітної плати інженера в розмірі 10000 грн з коефіцієнтом зайнятості 0.2, маємо:

$$C_{\text{г}} = 12 * 10000 * 0.2 = 24000 \text{ грн}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{зп}} = 24000 * (1 + 0.2) = 28800 \text{ грн}$$

Відрахування на соціальний внесок:

$$C_{\text{від}} = 28800 * 0.22 = 6336 \text{ грн}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн:

$$C_{\text{А}} = 1.15 * 0.25 * 25000 = 7187.5 \text{ грн}$$

Витрати на ремонт та профілактику можна підрахувати:

$$C_{\text{Р}} = 1.15 * 0.05 * 25000 = 1437.5 \text{ грн}$$

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\text{Т}_{\text{еф}} = (365 - 116 - 4) * 8 * 0.9 = 1764 \text{ годин}$$

Розрахуємо витрати на оплату електроенергії (з урахуванням ПДВ):

$$\text{С}_{\text{ел}} = 1764 * 0.3 * 1.46255 * 1.2 \ = 928.78 \text{ грн}$$

Накладні витрати рахуються наступним чином:

$$С_{н} = 25000 * 0.67 = 16750 \text{ грн}$$

Річні експлуатаційні витрати:

$$С_{екс} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н} =$$
$$= 28800 + 6336 \ + 7187.7 + 1437.5 + 928.78 + 16750 =$$
$$= 61519.98 \text{ грн}$$

Собівартість однієї машино-години ЕОМ становитиме:

$$С_{м-г} = 61519.98 \ / \ 1764 = 34.88 \text{ грн/год}$$

Витрати на оплату машинного часу складають в залежності від варіанту:

$$С_{м1} = 34.88 * 889.92 = 31036.2 \text{ грн},$$
$$С_{м2} = 34.88 * 1321.648 = 46099.01 \text{ грн}$$

Накладні витрати складають 67% від заробітної плати:

$$С_{н1} = 63540.29 * 0.67 = 42572 \text{ грн},$$
$$С_{н2} = 94365.67 * 0.67 = 63225 \text{ грн}$$

Таким чином, вартість розробки ПП та проведення дослідів складає:

$$С_{ПП1} = 63549.29 + 13978.86 + 31036.2 + 42572 = 151136.35 \text{ грн}$$
$$С_{ПП2} = 94365.67 + 18873.13 + 46099.01 + 63225 = 222562.81 \text{ грн},$$
$$С_{ПП2} = 94365.67 + 18873.13 + 46099.01 + 63225 = 222562.81 \text{ грн}$$

4.4 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулами:

$$K_{TEP1} = \frac{6.79}{151136.35} = 4.493 * 10^{-5},$$

$$K_{TEP2} = 5/222562.81 = 2.247 * 10^{-5}$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{TEP} = 4.493 * 10^{-5}$.

Висновки

В даному розділі був проведений функціонально-вартісний аналіз системи для класифікації міських звуків.

У першій частині розділу було проведено дослідження ПП з технічної точки зору. У другій частині отримані з першої варіанти реалізації були обґрунтовані з економічної точки зору.

Після виконання аналізу ПП, було виявлено, що оптимальним є перший варіант реалізації з показником техніко-економічного рівня якості $K_{TEP} = 4.493 * 10^{-5}$. Обраний варіант реалізації програмного продукту має такі параметри:

- мова програмування - Python;
- бібліотека для побудови моделі - Keras;
- середовище розробки - PyCharm.

Даний варіант виконання програмного продукту відповідає усім поставленим вимогам та може бути розроблений відносно швидко.

ВИСНОВКИ

Однією з найпоширеніших проблем класифікації звуків, яку слід вирішити останнім часом, є проблема класифікація міських звуків, адже процес стрімкої урбанізації світу розпочався не так давно і ми все більше і більше не можемо уявити собі життя без мегаполісів. Побудовані системи, що можуть класифікувати міські звуки, можна буде в майбутньому використовувати, наприклад, в якості аудіо підсистем для функціонування розумних будинків.

На сьогоднішній день найпопулярніший і найефективніший підхід до вирішення проблеми класифікації звуків, приймаючи до уваги, що задача зводиться до задачі класифікації зображень, є використання глибоких згорткових нейронних мереж.

При розробці власної системи класифікації міських звуків був використаний набір даних міських звуків UrbanSounds8K, який був знайдений на відкритій платформі Каггл. Для того, щоб класифікувати звуки, було натренеровано 5 згорткових нейронних мереж протягом 30 епох кожна, які потім були об'єднані в ансамбль за беггінг-принципом. Модель оцінювалася за декількома метриками та показала наступну трійку результатів: 0.93 за accuracy, 0.93 за micro f1 та 0.94 за macro f1, що дає змогу описувати ансамбль тільки за одним показником: 0.933, що є середнім між цими трьома показниками та за яким можна інтерпретувати точність ансамблю за будь-якою класичною метрикою класифікації.

Ці результати виявилися достатньо хорошими, щоб побудувати десктопну додаток-систему з інтуїтивно зрозумілим інтерфейсом на англійській мові, щоб користувач міг взаємодіяти з моделлю та класифікувати потрібні йому аудіо файли.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Щербакова Е. В 1950 году в мире было 2 мега-города с населением не менее 10 миллионов жителей, в 2011 году их стало 23, а в 2025 году может быть 37. *Демоскоп Weekly*. 2013. Выпуск №551-552. URL: http://www.demoscope.ru/weekly/2013/0551/barom03.php (дата звернення: 15.04.2020).
2. Portet, F., Vacher, M., Golanski, C. et al. Design and evaluation of a smart home voice interface for the elderly: acceptability and objection aspects. *Pers Ubiquit Comput.* 2013. Vol.17. P. 127–144. URL: https://doi.org/10.1007/s00779-011-0470-5 (Last accessed: 20.04.2020).
3. Manfred Schroeder. Fractals, Choas, Power Laws. New York: W.H. Freeman, 1991. 430 p.
4. Radford M. Neal. Bayesian Learning for Neural Networks. New York: Springer-Verlag New York, Inc., 1996. 117 p.
5. Gerhard, David. Audio signal classification: History and current techniques. Regina: Department of Computer Science, University of Regina, 2003. 37 p.
6. Analog to Digital Conversion. *GeeksForGeeks*. URL: https://www.geeksforgeeks.org/analog-to-digital-conversion/ (Last accessed: 13.04.2020).
7. Brian C. M. Moore. Hearing. Toronto: Academic Press, 1995. 463 p.
8. Saunders, John. Real-time discrimination of broadcast speech/music. 1996 *IEEE:* International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, Atlanta, GA, USA, 9 May 1996. Vol. 2. P. 993-996.
9. Erling Wold, Thom Blum, Douglas Keislar, and James Wheaton. Content-based classification, search and retrieval of audio. *IEEE: MultiMedia*, 1996. P. 27–37.
10. Eric Scheirer and Malcolm Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. *IEEE:* International Conference on

Acoustics, Speech and Signal Processing, Munich, Germany, 21-24 April 1997. Vol. II. P. 1331–1334.

11. Beth Logan. Mel Frequency Cepstral Coefficients for Music Modeling. *Ismir*. Vol. 270. 2000. 13 p.
12. Маркина Ю.Ю., Белов Ю.С. Кепстральные коэффициенты как необходимая характеристика процесса создания системы имитации голоса человека с помощью методов глубокого обучения. *Международный студенческий научный вестник*: 2018, выпуск №1. URL: http://www.eduherald.ru/ru/article/view?id=18125 (дата обращения: 25.04.2020)
13. Уэно Х., Исидзука М. Представление и использование знаний. *Пер. с яп.* Мир. 1989. 220 с.
14. Oliver Theobald. Machine Learning For Absolute Beginners: A Plain English Introduction. *Scatterplot Press*. 2017. P. 157.
15. Bengio, Yoshua, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence 35.8.* 7 March 2013. P. 1798-1828.
16. Guoqiang Zhong, Li-NaWang, Xiao Ling, Junyu Dong. An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*. 2016. Vol. 2. Issue 4. P. 265-278
17. Sandro Skansi. Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence. Zagreb, Croatia. Springer, 2018. 189 p.
18. Palm, Günther, and Ad Aertsen. Brain Theory: Proceedings of the First Trieste Meeting on Brain Theory. Trieste, Italy: Springer Science & Business Media, 1984. 257 p.
19. Luke B. Godfrey, Michael S. Gashler. A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks. *7th International Joint Conference on Knowledge*

Discovery, Knowledge Engineering and Knowledge Management. Lisbon, Portugal, 12-14 November 2015. Vol. 1. P. 481-486.

20. Klebanov, Lev Borisovich, Svetlozar Todorov Rachev, and Frank J. Fabozzi. Robust and non-robust models in statistics. New York, USA: Nova Science Publishers, 2009. Chapter 2. P. 165-226.
21. Ciresan, Dan Claudiu, et al. Flexible, high performance convolutional neural networks for image classification. *Twenty-Second International Joint Conference on Artificial Intelligence.* 2011. Vol. 2. P. 1237-1242.
22. Krizhevsky, A., Sutskever, I., Hinton, G. E.. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems.* 2012. P. 1097-1105.
23. Gao, Bolin, and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*. 2017. 10 p.
24. Tetko, Igor V., David J. Livingstone, and Alexander I. Luik. Neural network studies. 1. Comparison of overfitting and overtraining. *Journal of chemical information and computer sciences 35.5.* 1995. P. 826-833.
25. Ho, Daniel, et al. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393.* 2019. 14 p.
26. Suykens, Johan AK, Marco Signoretto, and Andreas Argyriou, eds. Regularization, optimization, kernels, and support vector machines. New York: CRC Press. 2014. 525 p.
27. Srivastava, Nitish, et al. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*. June, 2014. P. 1929-1958.
28. Neal, Brady, et al. A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591.* 2018. 28 p.
29. Zhou, Zhi-Hua. Ensemble methods: foundations and algorithms. New York: CRC press, 2012. 23 p.

30. Schapire, Robert E. The Strength of Weak Learnability. *Machine Learning. 5.2*. 1990. P. 197–227.
31. Breiman, Leo. Bias, variance, and arcing classifiers. Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA, 1996. 25 p.
32. Emer, Eric. Boosting (adaboost algorithm). URL: http://www.datascienceassn.org/sites/default/files/Adaboost%20Algorithm.pdf.
33. Aytar, Y., Vondrick, C., & Torralba, A. Soundnet: Learning sound representations from unlabeled video. *Advances in neural information processing systems*. 2016. P. 892-900.
34. Sterling, A., Wilson, J., Lowe, S., & Lin, M. C. ISNN: Impact sound neural network for audio-visual object classification. *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018. P. 555-572.
35. Koh, C. Y., Chang, J. Y., Tai, C. L., Huang, D. Y., Hsieh, H. H., & Liu, Y. W. Bird sound classification using convolutional neural networks. *CLEF working notes*. 2019. 10 p.
36. Salamon, J., & Bello, J. P. Unsupervised feature learning for urban sound classification. *2015 IEEE:* International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, QLD, Australia, 19-24 April 2015. P. 171-175
37. Salamon, J., Jacoby, C., & Bello, J. P. A dataset and taxonomy for urban sound research. Proceedings of the 22nd ACM international conference on Multimedia, Orlando, USA, November 2014. P. 1041-1044.
38. What's the deal with Accuracy, Precision, Recall and F1? *TowardsDataScience*. URL: https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021.
39. Özgür, A., Özgür, L., & Güngör, T. Text categorization with class-based and corpus-based keyword selection. International Symposium on Computer and Information Sciences. Springer, Berlin, Heidelberg. October, 2005. P. 606-615.

Додаток А Лістинг програмного модулю

Модуль model.py:

```
import os
import gc
import json
from glob import glob
import configparser

import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from PyQt5 import QtCore
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Flatten, Dropout
from keras.models import Sequential, model_from_json
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adagrad, Adam, SGD, RMSprop, Nadam
from path import Path
from sklearn.metrics import accuracy_score, classification_report, f1_score

#_translate = QtCore.QCoreApplication.translate
```

```
def save_dict(dictionary, path):
    j = json.dumps(dictionary)
    f = open(path, 'w')
    f.write(j)
    f.close()


def load_dict(path):
    with open(path, 'r') as f:
        return json.load(f)


def append_ext(fn):
    return fn.split('.')[0] + ".png"


def create_spectrogram(filename, fig_size, name=None, path=None):
    plt.interactive(False)
    clip, sample_rate = librosa.load(filename, sr=None)
    fig = plt.figure(figsize=fig_size)
    ax = fig.add_subplot(111)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    ax.set_frame_on(False)
    S = librosa.feature.melspectrogram(y=clip, sr=sample_rate)
    librosa.display.specshow(librosa.power_to_db(S, ref=np.max))
    if path is not None:
        name_folder = path
        filename = Path(name_folder + name + '.png')
    else:
```

```
        name_folder = 'test/test.png'
        filename = Path(name_folder)
    plt.savefig(filename, dpi=400, bbox_inches='tight', pad_inches=0)
    plt.close()
    fig.clf()
    plt.close(fig)
    plt.close('all')
    del filename, name, clip, sample_rate, fig, ax, S


class ConvolutionalNeuralNetwork:
    def __init__(self):
        conf = configparser.ConfigParser()
        conf.read('config.ini')

        self.y_true = None
        self.labels_dictionary = None

        self.model = None
        self.train_generator = None
        self.valid_generator = None
        self.test_generator = None

        self.STEP_SIZE_TRAIN = None
        self.STEP_SIZE_VALID = None
        self.STEP_SIZE_TEST = None
        self.batch_size = int(conf['Model_information']['batch_size'])

        self.x_col = conf['User_information']['x_col']
        self.y_col = conf['User_information']['y_col']
```

```
        self.y_col_class = conf['User_information']['y_col_class']
        x1 =
int(conf['Model_information']['input_shape'].split('(')[1].split(')')[0].split(',')[0])
        x2 =
int(conf['Model_information']['input_shape'].split('(')[1].split(')')[0].split(',')[1])
        x3 =
int(conf['Model_information']['input_shape'].split('(')[1].split(')')[0].split(',')[2])
        self.input_shape = tuple((x1, x2, x3))
        x4 =
float(conf['Model_information']['fig_size'].split('(')[1].split(')')[0].split(',')[0])
        x5 =
float(conf['Model_information']['fig_size'].split('(')[1].split(')')[0].split(',')[1])
        self.fig_size = tuple((x4, x5))

        self.epochs = int(conf['Model_information']['epochs'])
        self.n_classes = int(conf['User_information']['n_classes'])
        self.classes = None

        self.folds = int(conf['Number_of_models_in_ensemble']['folds'])
        self.folds_for_dataset =
int(conf['Number_of_models_in_ensemble']['folds_for_dataset'])
        self.test_fold = int(conf['Test_fold']['test_fold'])

        self.path = conf['User_information']['path']
        self.path_metadata = conf['User_information']['path_metadata']
        self.path_json = conf['Model_path_information']['path_json']
        self.path_h5 = conf['Model_path_information']['path_h5']
        self.path_dataset = conf['Model_path_information']['path_dataset']
        self.path_dict = conf['Model_path_information']['path_dict']
```

```
  def model_initialize(self):
      self.model = Sequential()
      self.model.add(Conv2D(128, (3, 3), padding="same", activation="tanh",
input_shape=self.input_shape))
      self.model.add(MaxPooling2D(pool_size=(2, 2)))
      self.model.add(Dropout(0.25))
      self.model.add(Conv2D(128, (3, 3), padding="same", activation="tanh"))
      self.model.add(MaxPooling2D(pool_size=(2, 2)))
      self.model.add(Dropout(0.25))
      self.model.add(Conv2D(64, (3, 3), padding="same", activation="tanh"))
      self.model.add(MaxPooling2D(pool_size=(2, 2)))
      self.model.add(Dropout(0.25))
      self.model.add(Conv2D(64, (3, 3), padding="same", activation="tanh"))
      self.model.add(MaxPooling2D(pool_size=(2, 2)))
      self.model.add(Dropout(0.25))
      self.model.add(Flatten())
      self.model.add(Dense(512, activation="tanh"))
      self.model.add(Dropout(0.25))
      self.model.add(Dense(512, activation="tanh"))
      self.model.add(Dropout(0.25))
      self.model.add(Dense(256, activation="tanh"))
      self.model.add(Dropout(0.25))
      self.model.add(Dense(256, activation='tanh'))
      self.model.add(Dense(self.n_classes, activation="softmax"))

      self.model.compile(optimizer=SGD(learning_rate=0.005, decay=1e-6,
momentum=0.9, nesterov=True),
                         loss="categorical_crossentropy", metrics=['accuracy'])
      self.model.summary()
```

```
def create_dataset(self):
    for k in range(1, self.folds_for_dataset + 1):
        p = Path(self.path + str(k) + "/*")
        g = glob(p)
        data_dir = np.array(g)
        for file in data_dir:
            filename, name = file, file.split('/')[-1].split('\\')[-1].split('.')[0]
            create_spectrogram(filename, self.fig_size, name, self.path_dataset)
        gc.collect()
    del data_dir, filename, name

def prepare_dataset(self):
    df = pd.read_csv(self.path_metadata, dtype=str)
    df = df.sample(frac=1).reset_index(drop=True)
    df[self.x_col] = df[self.x_col].apply(append_ext)
    self.classes = sorted(list(df[self.y_col].unique()))
    return df

def preprocess_train(self, dataset):
    divider = (self.folds - 1) * len(dataset) / self.folds
    train_df = dataset.loc[:divider-1, :]
    train_df = train_df.sample(frac=1).reset_index(drop=True)
    train_df = train_df.loc[:0.5*len(train_df)]

    datagen = ImageDataGenerator(rescale=1. / 255., validation_split=0.25)

    self.train_generator = datagen.flow_from_dataframe(
        dataframe=train_df,
        directory=self.path_dataset,
        x_col=self.x_col,
```

```
            y_col=self.y_col,
            subset='training',
            batch_size=self.batch_size,
            seed=42,
            shuffle=False,
            class_mode='categorical',
            target_size=(64, 64),
            classes=self.classes)

        self.valid_generator = datagen.flow_from_dataframe(
            dataframe=train_df,
            directory=self.path_dataset,
            x_col=self.x_col,
            y_col=self.y_col,
            subset='validation',
            batch_size=self.batch_size,
            seed=42,
            shuffle=False,
            class_mode='categorical',
            target_size=(64, 64),
            classes=self.classes)

        self.STEP_SIZE_TRAIN = self.train_generator.n //
self.train_generator.batch_size
        self.STEP_SIZE_VALID = self.valid_generator.n //
self.valid_generator.batch_size

    def preprocess_test(self, mod, dataset):
        if not mod:
            test_df = pd.read_csv(self.path_metadata, dtype=str)
```

```
            test_df = test_df[test_df.fold == str(self.test_fold)]
            test_df[self.x_col] = test_df[self.x_col].apply(append_ext)
        else:
            divider = (self.folds - 1) * len(dataset) / self.folds
            test_df = dataset.loc[divider:, :]

        n_labels = self.batch_size * int(len(test_df) / self.batch_size)
        self.y_true = list(test_df[self.y_col][:n_labels])
        self.y_true = [int(i) for i in self.y_true]

        test_datagen = ImageDataGenerator(rescale=1. / 255.)

        self.test_generator = test_datagen.flow_from_dataframe(
            dataframe=test_df,
            directory=self.path_dataset,
            x_col=self.x_col,
            y_col=None,
            batch_size=self.batch_size,
            seed=42,
            shuffle=False,
            class_mode=None,
            target_size=(64, 64))

        self.STEP_SIZE_TEST = self.test_generator.n // self.test_generator.batch_size

    def preprocess_predict(self, path, filename):
        df = pd.DataFrame(data=['test.png'], columns=['predict'])
        file = path + filename
        create_spectrogram(file, self.fig_size)
```

```
        test_df = pd.read_csv(self.path_metadata, dtype=str)
        self.labels_dictionary = {i: test_df.loc[test_df.loc[test_df[self.y_col] ==
i].index[0], self.y_col_class]
                                  for i in np.unique(test_df[self.y_col])}

        datagen = ImageDataGenerator(rescale=1. / 255.)

        generator = datagen.flow_from_dataframe(
            dataframe=df,
            directory='test/',
            x_col='predict',
            y_col=None,
            batch_size=1,
            seed=42,
            shuffle=False,
            class_mode=None,
            target_size=(64, 64))

        step_size = generator.n // generator.batch_size
        return generator, step_size

    def train(self, k):
        callbacks = ModelCheckpoint(str(k) + self.path_h5, save_best_only=True,
save_weights_only=True,
                                    monitor='val_accuracy', mode='max')

        self.model.fit_generator(generator=self.train_generator,
                                 steps_per_epoch=self.STEP_SIZE_TRAIN,
                                 validation_data=self.valid_generator,
                                 validation_steps=self.STEP_SIZE_VALID,
```

```
                    epochs=self.epochs,
                    callbacks=[callbacks])

        self.model.evaluate_generator(generator=self.valid_generator,
                                      steps=self.STEP_SIZE_VALID)

        save_dict(self.train_generator.class_indices, str(k) + self.path_dict)

    def test(self):
        self.test_generator.reset()

        predicted_class_indices = [[0 for k in range(self.n_classes)] for j in
range(len(self.y_true))]
        for k in range(0, self.folds - 1):
            self.load(k)
            pred = self.model.predict_generator(self.test_generator,
steps=self.STEP_SIZE_TEST, verbose=1)
            labels = load_dict(str(k) + self.path_dict)
            labels = [int(k) for k in labels]
            labels = np.argsort(labels)
            for i in pred:
                i = i[labels]
            for i in range(len(predicted_class_indices)):
                for j in range(self.n_classes):
                    predicted_class_indices[i][j] += pred[i][j]

        predicted_class_indices = np.argmax(predicted_class_indices, axis=1)

        accuracy = accuracy_score(self.y_true, predicted_class_indices)
        print(classification_report(self.y_true, predicted_class_indices))
```

```
        print(f1_score(self.y_true, predicted_class_indices, average='micro'))
        print(f1_score(self.y_true, predicted_class_indices, average='macro'))
        return accuracy

    def predict(self, generator, step_size):
        generator.reset()

        predicted_class_indices = [[0 for k in range(len(self.labels_dictionary))] for j
in range(len(generator))]
        for k in range(0, self.folds - 1):
            self.load(k)
            pred = self.model.predict_generator(generator, steps=step_size)
            labels = load_dict(str(k) + self.path_dict)
            labels = [int(k) for k in labels]
            labels = np.argsort(labels)
            for i in pred:
                i = i[labels]
            for i in range(len(predicted_class_indices)):
                for j in range(len(self.labels_dictionary)):
                    predicted_class_indices[i][j] += pred[i][j]

        predicted_class_indices = np.argmax(predicted_class_indices, axis=1)

        prediction = [self.labels_dictionary[str(k)] for k in predicted_class_indices]
        if len(prediction) == 1:
            prediction = prediction[0]
        return prediction

    def save(self, k):
        k = int(k)
```

```
        model_json = self.model.to_json()
        with open(str(k) + self.path_json, "w") as json_file:
            json_file.write(model_json)
        self.model.save_weights(str(k) + self.path_h5)

    def load(self, k):
        json_file = open(str(k) + self.path_json, 'r')
        loaded_model_json = json_file.read()
        json_file.close()
        self.model = model_from_json(loaded_model_json)
        self.model.load_weights(str(k) + self.path_h5)
        self.model.compile(optimizer='adam', loss="categorical_crossentropy",
metrics=['accuracy'])



class NN:
    def __init__(self, label):

        try:
            self.model = ConvolutionalNeuralNetwork()
            self.dataset = None
            self.label = label
            self.label.setText(_translate("MainWindow", "Preprocessing..."))
            print("Preprocessing...")
            self.label.show()
        except:
            self.label.setText(_translate("MainWindow", "Something went wrong"))

    def fit(self):
        try:
```

```
            self.model.create_dataset()
            self.label.setText(_translate("MainWindow", "Image dataset was created"))
            print("Image dataset was created")
        except:
            self.label.setText(_translate("MainWindow", "Something went wrong"))

    def train(self):
        try:
            self.dataset = self.model.prepare_dataset()
            for k in range(0, self.model.folds-1):
                self.model.preprocess_train(self.dataset)
                self.label.setText(_translate("MainWindow", "Data {} was
preprocessed".format(k)))
                print("Data {} was preprocessed".format(k))
                self.model.model_initialize()
                self.label.setText(_translate("MainWindow", "Model {} was
created".format(k)))
                print("Model {} was created".format(k))
                self.model.train(k)
                self.label.setText(_translate("MainWindow", "Model {} was
trained".format(k)))
                print("Model {} was trained".format(k))
                self.model.save(k)
                self.label.setText(_translate("MainWindow", "Model {} was saved to
disk".format(k)))
                print("Model {} was saved to disk".format(k))
            self.test(True, self.dataset)
        except:
            self.label.setText(_translate("MainWindow", "Something went wrong"))
```

```
  def test(self, mod=False, dataset=None):
      try:
          self.model.preprocess_test(mod, dataset)
          self.label.setText(_translate("MainWindow", "Data was preprocessed"))
          print("Data was preprocessed")
          accuracy = self.model.test()
          print(accuracy)
          self.label.setText(_translate("MainWindow", "Accuracy:
{}".format(round(accuracy, 3))))
      except:
          self.label.setText(_translate("MainWindow", "Something went wrong"))

  def predict(self, file_name, path):
      try:
          generator, step_size = self.model.preprocess_predict(path, file_name)
          prediction = self.model.predict(generator, step_size)
          self.label.setText(_translate("MainWindow", "It is the
{}".format(prediction)))
      except:
          self.label.setText(_translate("MainWindow", "Something went wrong"))


def fit(label):
  cnn = NN(label)
  cnn.fit()


def train(label):
  cnn = NN(label)
  cnn.train()
```

```
def test(label):
  cnn = NN(label)
  cnn.test()


def predict(label, file, path):
  cnn = NN(label)
  cnn.predict(file, path)
```

Модуль main.py:

```
import sys

from PyQt5.QtWidgets import QApplication, QDialog
from PyQt5 import QtCore

import model
from main_window import Ui_MainWindow


app = QApplication(sys.argv)
app.setApplicationName('Urban Sound Classification System')
_translate = QtCore.QCoreApplication.translate


class MainWindow(QDialog, Ui_MainWindow):
```

```
    def __init__(self, *args):
        super(MainWindow, self).__init__(*args)

        self.setupUi(self)
        self.pushButton_2.clicked.connect(self.click_switcher)
        self.pushButton_3.clicked.connect(self.click_extract)
        self.pushButton_4.clicked.connect(self.click_train)
        self.pushButton_5.clicked.connect(self.click_evaluate)
        self.pushButton_6.clicked.connect(self.click_result)

        self.switcher = True

    def click_result(self):
        self.label_5.setEnabled(True)
        file = self.lineEdit.text() if self.lineEdit.text() != '' else
self.lineEdit.placeholderText()
        path = self.lineEdit_2.text() if self.lineEdit_2.text() != '' else
self.lineEdit_2.placeholderText()
        model.predict(self.label_5, file, path)

    def click_extract(self):
        self.label_5.setEnabled(True)
        model.fit(self.label_5)

    def click_train(self):
        self.label_5.setEnabled(True)
        model.train(self.label_5)

    def click_evaluate(self):
        self.label_5.setEnabled(True)
```

```
        model.test(self.label_5)

    def click_switcher(self):
        self.label_5.setEnabled(False)
        self.label_5.setText(_translate("MainWindow", "Hi, researcher!"))
        self.pushButton_3.setEnabled(self.switcher)
        self.pushButton_4.setEnabled(self.switcher)
        self.pushButton_5.setEnabled(self.switcher)
        self.pushButton_6.setEnabled(not self.switcher)
        self.lineEdit.setEnabled(not self.switcher)
        self.lineEdit_2.setEnabled(not self.switcher)
        self.switcher = not self.switcher


form = MainWindow()
form.setWindowTitle('Urban Sound Classification System - Powered by Serhii
Smirnov')
form.show()
sys.exit(app.exec_())
```

Модуль main_window.py:

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
```

```
        MainWindow.resize(570, 764)
        MainWindow.setAutoFillBackground(False)
        MainWindow.setStyleSheet("image: url(background_image.jpg)")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setStyleSheet("image:url()")
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(340, 260, 211, 81))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        self.pushButton_2.setFont(font)
        self.pushButton_2.setStyleSheet("image:url();background-color: rgb(59, 203,
203);")
        self.pushButton_2.setObjectName("pushButton_2")
        self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_3.setEnabled(False)
        self.pushButton_3.setGeometry(QtCore.QRect(340, 350, 211, 81))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        self.pushButton_3.setFont(font)
        self.pushButton_3.setStyleSheet("image:url();background-color: rgb(196,
255, 247);")
        self.pushButton_3.setObjectName("pushButton_3")
        self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_4.setEnabled(False)
        self.pushButton_4.setGeometry(QtCore.QRect(340, 440, 211, 81))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
```

```
        font.setPointSize(12)
        self.pushButton_4.setFont(font)
        self.pushButton_4.setStyleSheet("image:url();background-color: rgb(196,
255, 247);")
        self.pushButton_4.setObjectName("pushButton_4")
        self.pushButton_5 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_5.setEnabled(False)
        self.pushButton_5.setGeometry(QtCore.QRect(340, 530, 211, 81))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        self.pushButton_5.setFont(font)
        self.pushButton_5.setStyleSheet("image:url();background-color: rgb(196,
255, 247);")
        self.pushButton_5.setObjectName("pushButton_5")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setEnabled(True)
        self.label.setGeometry(QtCore.QRect(180, 60, 221, 51))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(22)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setAcceptDrops(False)
        self.label.setAutoFillBackground(False)
        self.label.setStyleSheet("image:url();color: rgb(0, 0, 127);")
        self.label.setLineWidth(2)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setWordWrap(False)
```

```
        self.label.setObjectName("label")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(20, 480, 81, 31))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        self.label_3.setFont(font)
        self.label_3.setStyleSheet("image:url();background-color: rgb(59, 203,
203);")
        self.label_3.setAlignment(QtCore.Qt.AlignCenter)
        self.label_3.setObjectName("label_3")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(20, 430, 81, 31))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        self.label_4.setFont(font)
        self.label_4.setStyleSheet("image:url();background-color: rgb(59, 203,
203);")
        self.label_4.setAlignment(QtCore.Qt.AlignCenter)
        self.label_4.setObjectName("label_4")
        self.pushButton_6 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_6.setGeometry(QtCore.QRect(20, 530, 241, 81))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        self.pushButton_6.setFont(font)
        self.pushButton_6.setStyleSheet("image:url();background-color: rgb(59, 203,
203);")
        self.pushButton_6.setObjectName("pushButton_6")
```

```
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setEnabled(False)
        self.label_5.setGeometry(QtCore.QRect(20, 260, 241, 91))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(14)
        self.label_5.setFont(font)
        self.label_5.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label_5.setStyleSheet("image:url();background-color: rgb(196, 255,
247);")
        self.label_5.setAlignment(QtCore.Qt.AlignCenter)
        self.label_5.setObjectName("label_5")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setEnabled(True)
        self.label_2.setGeometry(QtCore.QRect(180, 110, 221, 41))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(22)
        font.setBold(True)
        font.setWeight(75)
        self.label_2.setFont(font)
        self.label_2.setAcceptDrops(False)
        self.label_2.setAutoFillBackground(False)
        self.label_2.setStyleSheet("image:url();color: rgb(0, 0, 127)")
        self.label_2.setLineWidth(2)
        self.label_2.setAlignment(QtCore.Qt.AlignCenter)
        self.label_2.setWordWrap(False)
        self.label_2.setObjectName("label_2")
        self.label_6 = QtWidgets.QLabel(self.centralwidget)
        self.label_6.setEnabled(True)
```

```
        self.label_6.setGeometry(QtCore.QRect(180, 150, 221, 51))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(22)
        font.setBold(True)
        font.setWeight(75)
        self.label_6.setFont(font)
        self.label_6.setAcceptDrops(False)
        self.label_6.setAutoFillBackground(False)
        self.label_6.setStyleSheet("image:url();color: rgb(0, 0, 127)")
        self.label_6.setLineWidth(2)
        self.label_6.setAlignment(QtCore.Qt.AlignCenter)
        self.label_6.setWordWrap(False)
        self.label_6.setObjectName("label_6")
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(100, 480, 161, 31))
        self.lineEdit.setText("")
        self.lineEdit.setObjectName("lineEdit")
        self.lineEdit.setPlaceholderText("7383-3-0-0.wav")
        self.lineEdit_2 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_2.setGeometry(QtCore.QRect(100, 430, 161, 31))
        self.lineEdit_2.setText("")
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.lineEdit_2.setPlaceholderText("urbansound8k/fold1/")

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
        MainWindow.setFixedSize(MainWindow.geometry().width(),
MainWindow.geometry().height())
```

```
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Urban Sound
Classification System"))
        self.pushButton_2.setText(_translate("MainWindow", "Switcher"))
        self.pushButton_3.setText(_translate("MainWindow", "Extract Features"))
        self.pushButton_4.setText(_translate("MainWindow", "Train"))
        self.pushButton_5.setText(_translate("MainWindow", "Evaluate"))
        self.label.setText(_translate("MainWindow", "Urban Sound"))
        self.label_3.setText(_translate("MainWindow", "File"))
        self.label_4.setText(_translate("MainWindow", "Path"))
        self.pushButton_6.setText(_translate("MainWindow", "Result"))
        self.label_5.setText(_translate("MainWindow", "Hi, researcher!"))
        self.label_2.setText(_translate("MainWindow", "Classification"))
        self.label_6.setText(_translate("MainWindow", "System"))
```

Модуль config.ini:

```
[User_information]
path = urbansound8k/fold
path_metadata = urbansound8k/UrbanSound8k.csv
x_col = slice_file_name
y_col = classID
y_col_class = class
n_classes = 10

[Model_path_information]
path_dataset = dataset/
```

path_json = model.json

path_h5 = model.h5

path_dict = labels.json

[Model_information]

input_shape = (64, 64, 3)

fig_size = (0.72, 0.72)

batch_size = 42

epochs = 30

[Number_of_models_in_ensemble]

folds = 6

folds_for_dataset = 10

[Test_fold]

test_fold = 1

Додаток Б Презентація

Актуальність

Методологія

Система класифікації міських звуків на основі глибоких нейронних мереж

Модель

Система

Висновки

Смірнов Сергій Сергійович, КА-61
Керівник: доц., к. ф.-м. наук Каніовська І.Ю.

Актуальність

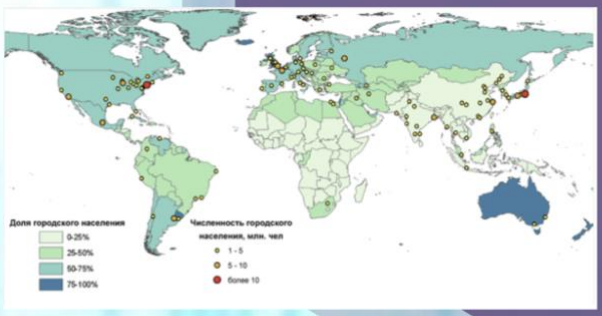
- Значення міста в житті сучасної людини стрімко зростає

Приклади використання - Smart House:

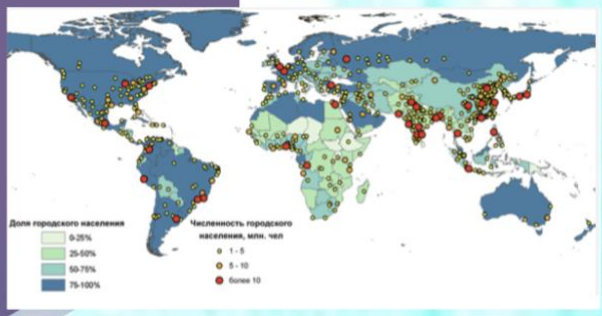
- "Голосне" місто
- Розумна безпека

Аргументація

Доля міського населення та міські агломерації



1960 рік



прогн. 2025 рік

3 / 13

Методологія

Чому глибоке навчання:

- Підхід баз знань
- Машинне навчання
- Глибоке навчання

Обробка звуків:

- Мел частотні кепстральні діаграми

Дані

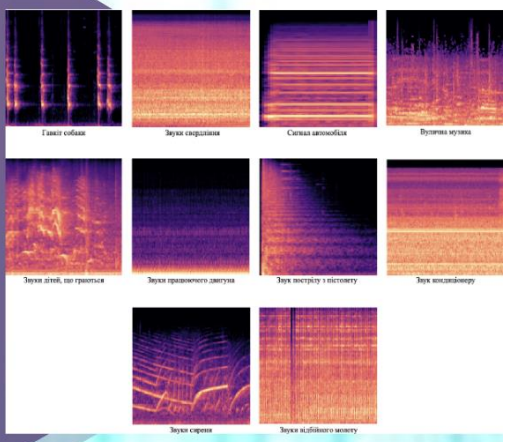
Подібні системи

4 / 13

Дані

UrbanSound8K набір:

- звук кондиціонеру
- сигнал автомобіля
- звуки дітей, що граються
- гавкіт собаки
- звуки свердління
- звук робочого двигуна
- звуки пострілу з пістолету
- звук відбійного молота
- звуки сирени
- вулична музика



5 / 13

Подібні системи

SoundNet:*

- розпізнавання, розділення та класифікація звуків на відео фреймі

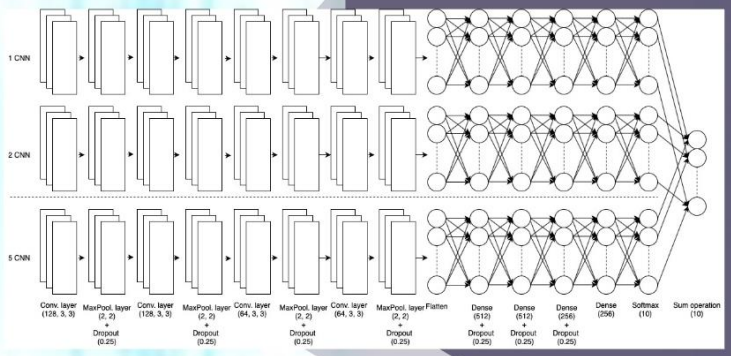
ISNN:**

- класифікація об'єктів на відеофреймі за допомогою звуків

* Aytar Y., Vondrick C., Torralba A. "SoundNet: Learning Sound Representations from Unlabeled Video"
** Sterling A., Wilson J., Lowe S., Lin M.C. "ISNN: Impact Sound Neural Network for Audio-Visual Object Classification"

6 / 13

Архітектура*



Повнозв'язний шар: $\mathbf{z}^l = W^l \mathbf{h}^{l-1}$

Згортковий шар: $\mathbf{z}^l = \mathbf{h}^{l-1} * W^l$

Функція активації: $y = tanh(c)$

Шар пуллінгу: $h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$

Цільова функція: $y = -\sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} log(\gamma_{ij}))$

Softmax: $S(c_i) = \frac{e^{c_i}}{\sum_j e^{c_j}}$

Вихід: $argmax(\{\sum_i^5 p_{ij}, j = 1 : 10\})$

* Смірнов С.: "Порівняльний аналіз нейронних мереж та їх ансамблів для задачі класифікації міських звуків"; стаття на V Міжнародній науково-практичній конференції "Відкриті системи, що еволюціонують" ТНУ ім. Вернандського

Система

Метрики

Демо

Метрики

клас звуку	$precision_i$	$recall_i$	$f1_i$
air_conditioner	0.84	0.98	0.91
car_horn	1	0.94	0.97
children_playing	0.91	0.91	0.91
dog_bark	0.96	0.91	0.93
drilling	0.97	0.9	0.93
engine_drilling	0.99	0.93	0.96
gun_shot	1	0.98	0.99
jackhammer	0.93	0.96	0.94
siren	0.86	0.99	0.92
street_music	0.95	0.84	0.89

accuracy	*micro f1*	*macro f1*
0.93	0.93	0.94

$$accuracy = \frac{\sum_i TP_i + TN_i}{b}$$

$$micro\ precision = \frac{TP}{TP + FP} = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)}$$

$$precision_i = \frac{TP_i}{TP_i + FP_i}$$

$$micro\ recall = \frac{TP}{TP + FN} = \frac{\sum_i TP_i}{\sum_i (TP_i + FN_i)}$$

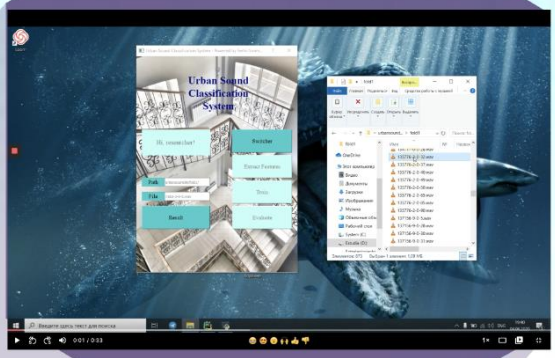
$$recall_i = \frac{TP_i}{TP_i + FN_i}$$

$$micro\ f1 = \frac{2 \cdot micro\ precision \cdot micro\ recall}{micro\ precision + micro\ recall}$$

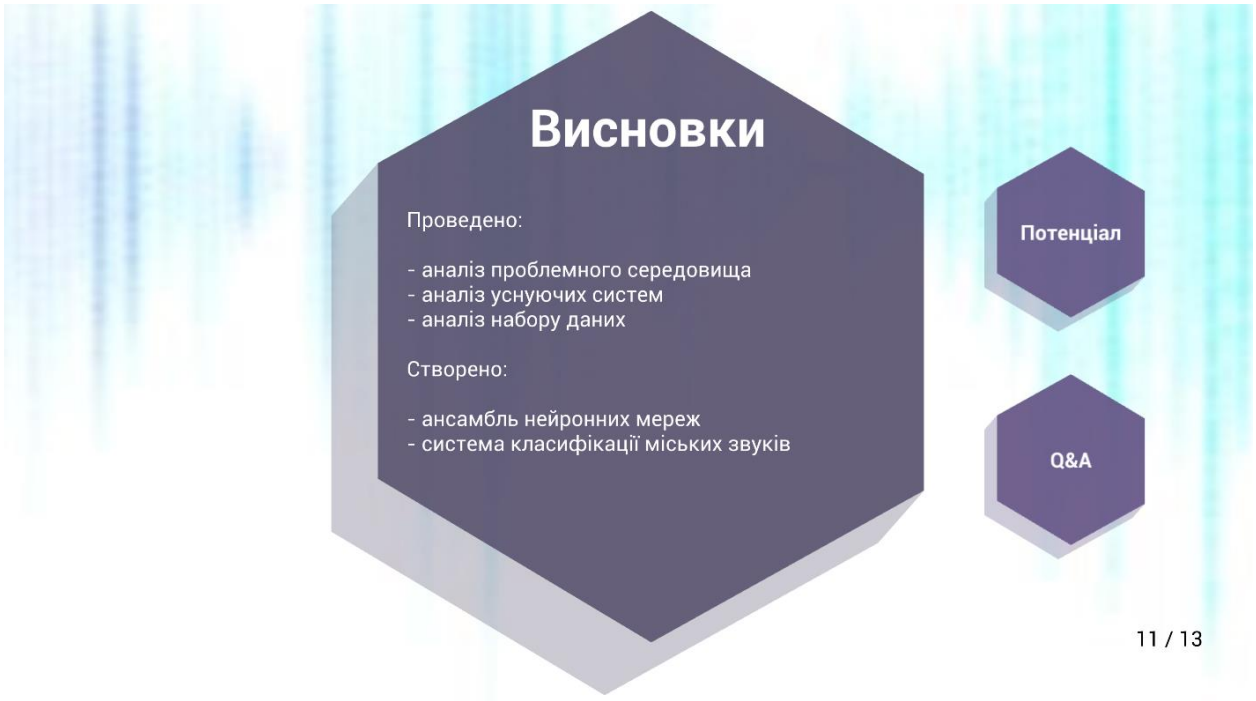
$$f1_i = \frac{2 \cdot precision_i \cdot recall_i}{precision_i + recall_i}$$

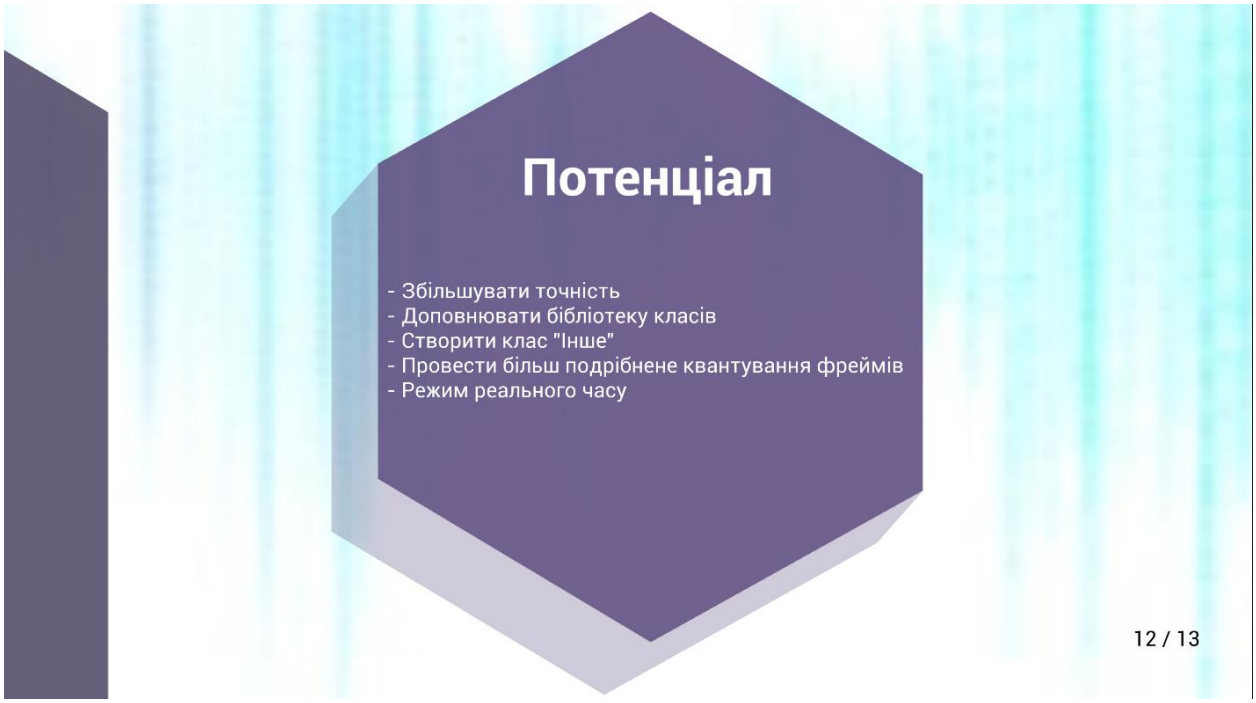
$$macro\ f1 = \frac{\sum_i f1_i}{M}$$

9 / 13



10 / 13


Висновки
Проведено:
- аналіз проблемного середовища
- аналіз уснуючих систем
- аналіз набору даних
Створено:
- ансамбль нейронних мереж
- система класифікації міських звуків
Потенціал
Q&A
11 / 13


Потенціал
- Збільшувати точність
- Доповнювати бібліотеку класів
- Створити клас "Інше"
- Провести більш подрібнене квантування фреймів
- Режим реального часу
12 / 13


Дякую за увагу